

AD-A137 320

A DISTRIBUTED OPERATING SYSTEM DESIGN AND
DICTIONARY/DIRECTORY FOR THE ST... (U) NAVAL POSTGRADUATE
SCHOOL MONTEREY CA N F SCHNEIDEWIND ET AL. NOV 83
NPS54-83-015 F/G 9/2

1/1

UNCLASSIFIED

NL

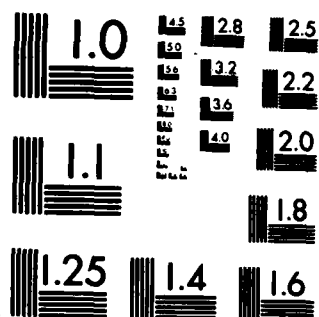
END

DATE

FILED

2 84

DTIC



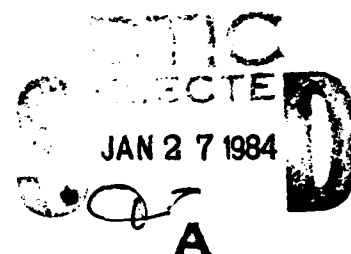
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A 137320

Report Number -- NPS54-83-015

NAVAL POSTGRADUATE SCHOOL

Monterey, California



A DISTRIBUTED OPERATING SYSTEM DESIGN AND
DICTIONARY/DIRECTORY FOR THE STOCK POINT
LOGISTICS INTEGRATED COMMUNICATIONS ENVIRONMENT

Norman F. Schneidewind
Daniel R. Dolk

November 1983

Final Report: 1 December 82 to 1 November 83

Approved for public release; distribution unlimited

Prepared for:
Fleet Material Support Office
Mechanicsburg, Pennsylvania

DTIC FILE COPY

84 01 27 030

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

David R. Schradly
Provost

The work reported herein was supported by the Fleet Material Support Office.

Reproduction of all or part of this report is authorized.


This report was prepared by:


NORMAN F. SCHNEIDEWIND
Professor of Computer Science


DANIEL R. DOLK, Assistant Professor
of Management Information Systems

Reviewed by:

Released by:


RICHARD S. ELSTER, Chairman
Department of Administrative Sciences


WILLIAM M. TOLLES
Dean of Research

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS-54-83-015	2. GOVT ACCESSION NO. AD-A137320	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A DISTRIBUTED OPERATING SYSTEM DESIGN AND DICTIONARY/DIRECTORY FOR THE STOCK POINT LOGISTICS INTEGRATED COMMUNICATIONS ENVIRONMENT		5. TYPE OF REPORT & PERIOD COVERED Final Report 1 DEC 82 - 1 NOV 83
7. AUTHOR(s) Norman F. Schneidewind Daniel R. Dolk		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Fleet Material Support Office Mechanicsburg, Pennsylvania		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N0036783PON3883
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE November 1983
		13. NUMBER OF PAGES 73
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Local Computer Networks, Distributed Computer Systems, Operating System Design, Dictionary/Directory Systems		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The problems and opportunities involved with designing and using distributed systems are discussed. This is followed by presenting a paradigm for the distributed system design process. The paradigm is then applied to the design of a distributed operating system for SPLICE. The major interface between user and operating system is provided by the dictionary/directory system.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

	Page number
°INTRODUCTION	1
°THE CHALLENGE OF DISTRIBUTED SYSTEMS	2
°SYSTEM ENTITIES	3
°DISTRIBUTED SYSTEM PARADIGM	5
°DISTRIBUTED SYSTEM DESIGN STRATEGY	30
°FUNCTIONAL MODULES	31
°NETWORK OPERATING SYSTEM DESIGN	36
°OPERATING SYSTEM DESIGN OBJECTIVES	37
°INTERPROCESS COMMUNICATION	40
°SCHEDULING	42
°DEADLOCK PREVENTION	44
°MEMORY (RAM) MANAGEMENT	46
°SHARED RESOURCES AND THE CRITICAL SECTION PROBLEM	48
°USER INTERFACE SPECIFICATIONS	53
-User calls	53
-Justification for SPLICE Dictionary/Directory System (DDS)	54
-Scope of DDS	56
-Contents and Logical Structure of DDS	58
-Distribution of the DDS	64
-Environmental Dependency	67
°COMPUTER ARCHITECTURE	69
°CONCLUSIONS AND RECOMMENDATIONS	71
°REFERENCES	76



LIST OF FIGURES

	Page number
1. Initial representation of objects in a system	9
2. Associating names and functions with objects	9
3a. Logical Ring	10
3b. Logical Bus	10
3c. Fully Connected Mesh	10
4a. Object Communication Protocol for DEMOS and ROSCOE	14
4b. Object Communication Protocol with Data and Control Links	14
4c. Object Communication Protocol of Figure 4b Implemented on a bus	14
5a. Interrupt Driven Object Communication Procedure	17
5b. Tandem Operating System Link Procedure	17
6. Object Addressing Using Names	21
7. Object Control Structure (Two Instances)	25
8. Object Invocation and Data Paths (Two Instances)	26
9. Message Assignment to Ports and Use of Kernel to Process Interrupts	35
10. Layered Operating System Design	38
11. Program Control Block Table	43
12. Ready List Data Structure	45
13. Buffer Allocation	47
14. Memory Map	49
15. Memory Map Table	50
16. Logical Structure Describing Data Resources	60
17. Logical Structure of Hardware Resources (from [Allen et al 1982])	62
18. Logical Structure for Software, Transaction, and Report Resources	62
19. Vector Interrupt Table	72

LIST OF TABLES

	Page number
0. Local Network Performance Metrics	28
1. Data Element Attributes (from Allen et al [1982])	58
2. File Entity Attributes	59
3. Selected Hardware Entities and Attributes	61
4. Selected Software Entities and Attributes	63
5. Document/Report Attributes	64

INTRODUCTION

This report is a follow on to a previous report [Schneidewind 82].

It is assumed that the reader is familiar with that report. The primary objectives of this report are the following:

- "Describe, specify and recommend an efficient, low overhead distributed operating system design for SPLICE.

- "Discuss the desirability of including a dictionary/directory system (DDS) as a major component of SPLICE.

The report is organized around the following major themes:

We begin with a discussion of the opportunities and problems involved with designing and using distributed systems. Since the field of distributed systems is still evolving and design procedures for developing these systems are not well understood, we provide a paradigm, or model, of the distributed system design process, with emphasis on an object-oriented network architecture approach. This model lays the groundwork for the proposed distributed operating system design which follows. In both the discussion of the paradigm and the operating system design, emphasis is placed on the use of message-based interprocess communication as the vehicle for achieving distributed control among the various functional modules which comprise SPLICE. After providing a functional specification for a distributed operating system, user interface specifications are provided, where the dictionary/directory system (DDS) constitutes the major component. The DDS is designed to provide information to both the user and system about data and resources (i.e. objects) which exist in SPLICE, and to provide names and addresses, so that these objects can be accessed in the network. The report concludes with a discussion of the advantages and

disadvantages of a distributed operating system design and a description of recommended steps for implementing a dictionary/directory system.

THE CHALLENGE OF DISTRIBUTED SYSTEMS

There are many interesting and challenging problems associated with the design of distributed systems. These problems arise primarily because a distributed system has many objects (e.g., software module, processor) which operate autonomously. This independent mode of operation is responsible for the increased effectiveness (e.g., increased speed of operation and reduced overhead) over a system with centralized control. However, once the many objects are turned loose, so to speak, without benefit of close supervision by the operating system, many problems of coordination arise. This 'laissez faire' operating environment presents challenges to the designer to find solutions to the following problems:

- *Procedures for interprocess and interprocessor communication and control
- *Naming and addressing of objects
- *Allocation of functions to modules and modules to processors
- *Distribution of data across nodes of the network
- *Specification of network topology
- *Methods for deadlock prevention, avoidance, detection and recovery
- *Procedures for recovery from system malfunctions
- *Specification of appropriate levels of performance

Although the list is incomplete, it is illustrative of the major design issue in distributed systems: achievement of coordination and

control of independent objects. Note that none of the problem areas are exclusive to distributed systems - they are just more complex and harder to solve in a distributed system. Also, observe that, in general, these problems are not confined to a single network of objects, such as a local network or long distance network. Rather, a multiplicity of networks may be used in an application or by an organization, so that solutions must be found in the context of internetting, that is the communication of data over diverse multiple networks. In the case of SPLICE, this involves 62 local networks interconnected by the Defense Data Network (DDN). A primary design objective of distributed systems is to allow the user to utilize objects independent of their geographical location and form of implementation [Watson 81 a]. This means, that except for a degradation in performance, it should be as convenient for a user at the Naval Supply Center (NSC) Oakland to access the Inventory Control Point (ICP) data base at Mechanicsburg, as it is to access the data base on the local computer network at Oakland. The remaining sections of this report describe the recommended design for a SPLICE distributed operating system and its associated data dictionary function, which provides support for naming and identifying objects in SPLICE.

SYSTEM ENTITIES

Many entities comprise a distributed system. These entities will be used to describe our distributed system design methodology and recommended distributed operating system design. Unfortunately, a variety of definitions can be found for these terms in the literature.

We define these terms below in accordance with their meaning in this report.

- *Object: anything intelligible or perceptible by the mind. A thing serving as the focus of attention. An object has a name and attributes. It could be, for example, a process, node, processor, module or resource. It is synonymous with entity.
- *Message: the unit of data transmitted between two objects.
- *Functional: designed for or adapted to a particular need or activity.
- *Functional Module (FM): an object which is functional (i.e., dedicated to a specific activity such as terminal management).
- *Components of an FM: Sub-Modules
 - Input (generalizable)
 - Output (generalizable)
 - Processing (unique)
- *Process: an FM in execution (Task is synonymous with process).
- *Requestor: an FM which requests a service.
- *Server: an FM which provides a service.
- *Resource: a consumable (e.g., printer paper) or non-consumable object (e.g., memory) which is used to support the operation of FMs.
- *Node: a physical element of a network (e.g. processor) which connects two physical communication links.
- *Logical (virtual) link: an imaginary communication path between two objects (FMs).
- *Physical link: a physical communication path between two nodes.
- *Logical Path: The desired route of a message, from sender to receiver, independent of the actual physical path used.

- °Physical Path: the actual route of a message from sender to receiver through various physical nodes and data links.
- °Connectivity: the number of object pairs which are directly connected by logical or physical links, depending upon circumstances, in a network.
- °Accessibility: the number of objects which a given object can reach directly (i.e., without going through intermediate objects)
- °Port: A data abstraction affiliated with an object. It has an address and a buffer.
- °Kernel: That part of an operating system which provides common services (e.g., memory allocation) in each processor.
- °Session: All of the activity (message exchange and processing) which takes place between two or more processes for the duration of a single task. (e.g. text edit or processing of a transaction file).
- °Layer: A partition in a computer network architecture model which is assigned a specific function (e.g., session layer).

In this report emphasis will be on the relationship between FMs as opposed to the activities of processes because many of the functions which must be executed are independent of whether the FM is executing (i.e., a process) as in, for example, when one FM sends a message to another. It is of no interest to the sending FM whether the receiving FM is executing or not.

DISTRIBUTED SYSTEM PARADIGM

Exact methods for designing distributed systems do not exist. Solutions to the problems identified in the previous section are being

pursued in various research programs. One consequence of this situation is that there are few examples of fully distributed systems in operation. An objective of the NPS research program for SPLICE is to advance our knowledge of distributed systems and to increase our understanding of how distributed systems should be designed in order to operate effectively. Our approach to achieving this objective, as described in this section, is: create a paradigm, or model, of the distributed system development process and use it as a guideline for designing the recommended SPLICE operating system, which is described in a later section. Furthermore, valuable insights into contemporary distributed system design practices - some convincing and others questionable - were obtained from a review of other research efforts and distributed operating system implementations. These design approaches influenced but did not dominate the creation of the paradigm specifications. Subsequently, the paradigm was compared with the various designs to determine their adequacy as models for designing distributed network operating systems (e.g., SPLICE). In addition to contributing to the understanding of distributed systems, an additional motivation for developing the paradigm was to focus attention on the logical properties of a distributed system. Too often, the determination of a distributed system architecture is driven by available hardware configurations (e.g., bus or ring) to the detriment of emphasizing good design procedure.

A paradigm for understanding how one could approach the design of distributed systems is to visualize the various phases of the development and growth of a distributed system from birth (conceptualization) to "adulthood" (operation). If, in our imagination, we could "simulate"

the growth of a system from its birth and observe how the system attains its attributes as it develops from the embryonic conceptualization stage into a mature architecture, the record of the maturation process could serve to identify those attributes and attribute relationships of a system which are critical to its effective operation. This approach allows us to understand how the objects of the system should be nurtured and molded in order to have the desired properties when the system reaches maturity. In particular, we will be concerned with the properties of a system which allow a high degree of decentralization of control - a distributed system [Jensen 81]. We would expect decentralization of control to lead to low system overhead which, in turn, should result in high speed performance. We should expect nothing less from a distributed system. For, if performance cannot be improved by using a distributed approach, why bother with the additional complexity which is incurred when state information and communication functions are distributed? That is to say, the easiest control system to design and implement is the classical operating system structure, where control is centralized in the supervisor and all requests for service and resource allocation decisions must be processed by the supervisor. We must be careful, when specifying the architecture of a distributed system, not to fall into the trap of re-introducing much of the rococo of centralized designs; otherwise, the goal of enhanced performance will be defeated. Although our focus will be on distributed system design, it should be noted that the methodology would apply equally well to non-distributed system design.

Conceptualization of a Distributed System

We begin the process of conceptualizing a system by identifying the objects of the system. An object is an abstract representation of a system entity such as a process, module or resource. One definition of an object is that it is an incarnation of a resource [Wulf 74]. This method of representation has the great advantage of generality; it can represent any logical system (e.g., distributed or centralized) structure, independent of the physical implementation of the system [Watson 81 b]. As a means of illustrating the evolution of a system, we show a series of figures which chronicle the development of our model system. Figure 1 shows the first stage - giving birth to general objects, where no attributes or communication paths among the objects have been assigned.

Next, we make these general objects specific by assigning attributes which are representative of their function. Our objects have a name (e.g., TM) and perform a single major function (e.g., terminal management). Three objects of SPLICE have been created by assigning names and functions as shown in Figure 2.

Object Communication Requirements

The third stage in the evolution of the objects into a system is for the objects to acquire the ability to communicate. This is accomplished by connecting the objects with logical links as shown in Figure 3. A logical link is defined as a representation of the capability to communicate between two objects. It indicates nothing about the actual physical connection topology which may be employed, once the hardware configuration has been defined. If four objects are involved, as shown in Figure 3 a, one possible logical interconnection scheme is the ring, where direction of message flow is shown as

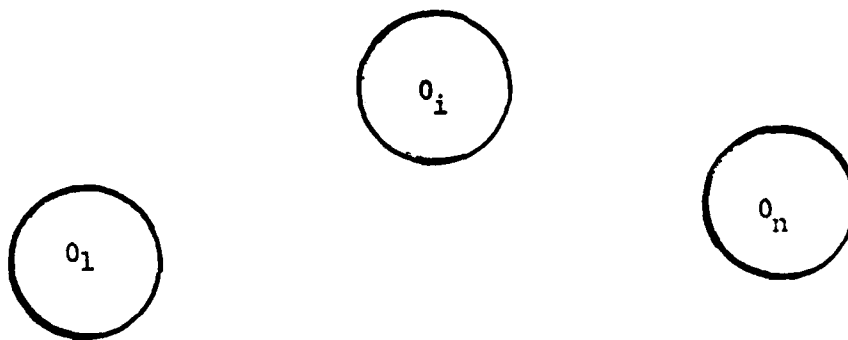


Figure 1 ... Initial representation of objects in a system

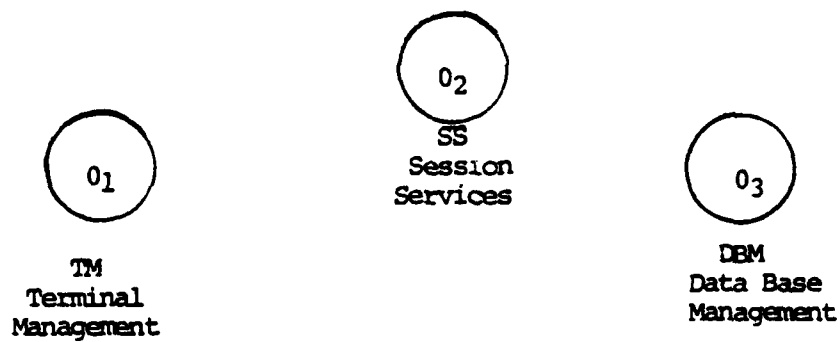


Figure 2 ... Associating names and functions with objects

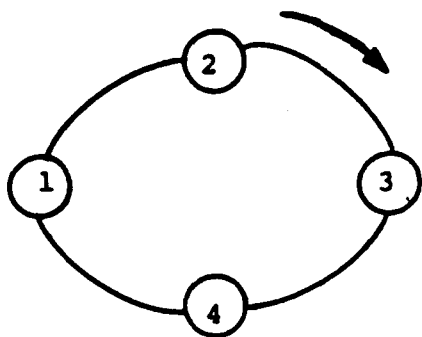


Figure 3a ... Logical Ring

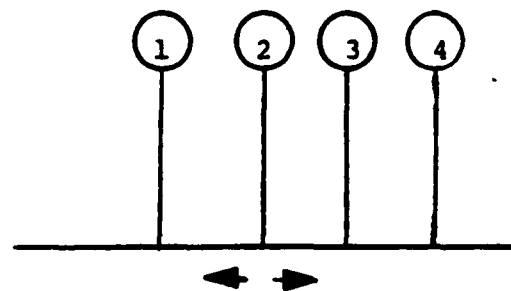


Figure 3b ... Logical Bus

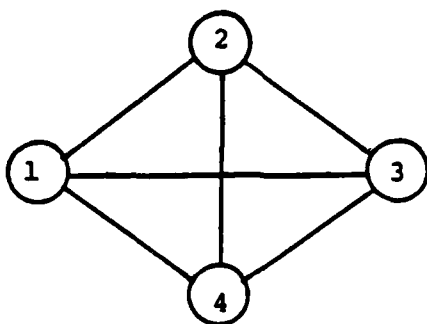


Figure 3c ... Fully Connected Mesh

clockwise. From a logical message processing standpoint this arrangement has two disadvantages:

(1) In order for an object to communicate with a non-adjacent object, the message must flow through and be handled by one or more nonaddressed objects (e.g., a message originating at 2 and addressed to 4, must flow through 3) and (2) a broadcast message (i.e., a message addressed to all objects) must flow through all objects before the communication is complete (e.g., a broadcast message originating at 1, must flow through 2, 3 and 4). A message should not flow through an object if it adds nothing of value to the message. In contrast, a message sent on the logical bus, shown in Figure 3b., is transmitted simultaneously to all objects on the bus, without the message having to travel through the objects. Thus, bus transmission logic is inherently broadcast mode. Since a bus topology is very general and flexible (i.e., a message can be sent to any or all objects without handling and processing by intermediaries), it provides a convenient design model for representing a local network type of distributed system. We note, in passing, that a fully connected mesh topology, as shown in Figure 3c, could be used for the model, since it provides a direct path from each object to every other object. However, in a sizeable system, this topological representation becomes very cluttered. More important, the number of links in a fully connected mesh is $n(n-1)/2$, or on the order of n^2 , where n is the number of objects.

Unfortunately, we must point out that, although the bus topology provides a clean logical representation, this model does not necessarily map into the physical world as the topology with the best performance. The reason is that, in addition to being a broadcast medium, a bus is

inherently a contention access system. Because of this characteristic and depending on the details of the hardware design, two problems could arise which could render the bus slower than the ring: (1) the bus is in use when a node (the physical analog of an object) wants to transmit and (2) a node's transmission collides (i.e., overlaps) with that of another node, causing the transmission to be aborted and rescheduled for a later, randomly determined time.

Interestingly, the matter of communication among objects brings to the fore one of the most controversial issues in the area of local network technology: bus versus ring topology [Pevovar 82, Parker 83]. A basic characteristic of a bus is the physical ability to provide broadcast communication (i.e., every node can receive the sending node's message essentially simultaneously). The maximum delay is the negligible signal propagation time over the length of the bus. A ring, on the other hand, does not possess this physical capability because a message must be serially transmitted around the ring. However, it must be noted that logically the equivalent of a broadcast transmission can be achieved by the sender using an address code which will cause every node to read the message, as it circulates in the ring. Whether a bus or ring topology does this faster or with greater throughput depends significantly on the load [Nadkarni 83, Saltzer 81, Salwen 83, Stuck 83]. With light load, there is less delay on a contention bus, relative to a token ring, because there is high probability that a node will be able to transmit its message immediately on the bus; also, there is no waiting for receipt of a token, as there is on a token ring. Conversely, at high loads, contention is so great on a bus that the probability is high that a node attempting to transmit will encounter

congestion. Indeed, there is no upper bound on delay time on a contention bus, whereas delay time is bounded for a token ring, because a node will be guaranteed to receive the right to transmit, via the receipt of the token, within finite time.

Object Protocols

Once the objects of the distributed system have been 'given' the ability to communicate, the manner in which they are to communicate - the protocols - must be decided. The design of a protocol involves many issues concerning procedures for communication, such as the following.

*Degree of formality in arranging for communication: Is it possible to simply send a message from O_i to O_j or is it necessary for O_i to first signal its intent to communicate and for O_j to agree to accept messages from O_i (i.e., handshaking)?

*Message communication procedure: A useful model for message communication between objects is the use of simplex (one-way) logical links for request and reply control functions and a full duplex (two-way simultaneous) logical link for data, similar to the link concept used in the DEMOS operating system for the CRAY-1 computer [Baskett 77] and in the ROSCOE Distributed Operating System at the University of Wisconsin [Solomon 79]. An illustration of this model is shown in Figure 4a. Although it is useful, it is unnecessarily complicated for efficient system performance and not entirely representative of the general case of object communication. This representation tends to artificially partition objects into sources of requests and replies and into requesters and servers. The Tandem Computer Corporation operating system uses a similar representation (see Figure 5b) [Bartlett 78].

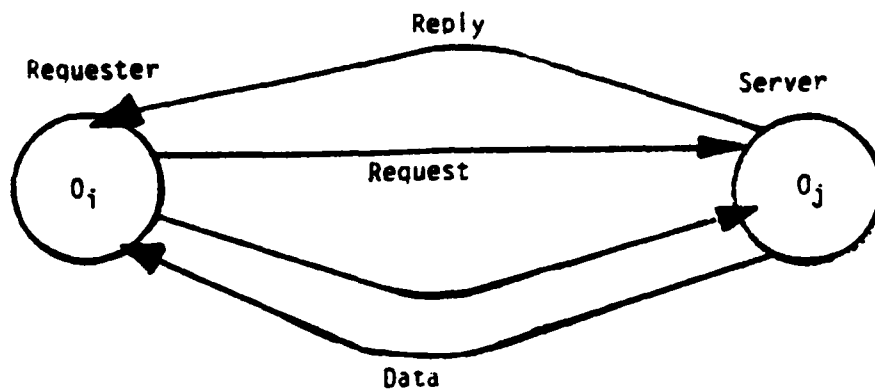
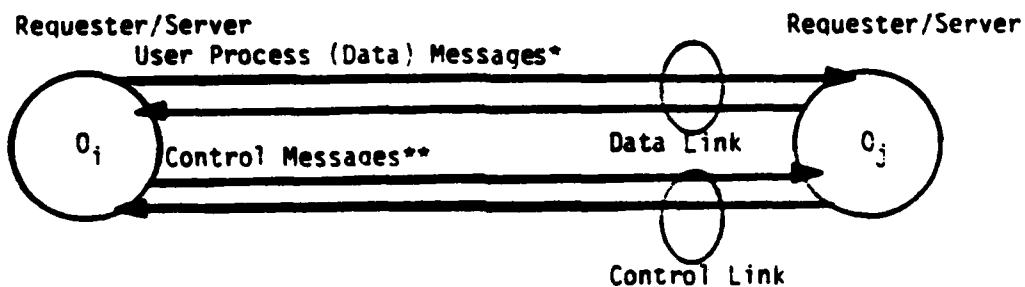


Figure 4a ... Object Communication Protocol for DEMOS and ROSCOE



*Requests, replies and data

**E.g., acknowledgements, error messages, recovery messages etc.

Figure 4b ... Object Communication Protocol with Data and Control Links

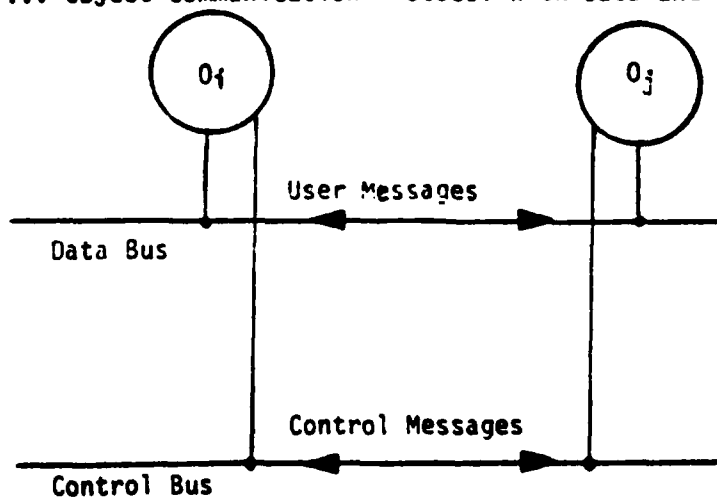


Figure 4c ... Object Communication Protocol of Figure 4b Implemented on a Bus

In general, both requests and replies can emanate from the same object. Furthermore, an object may be a server in one instance and a requester in another. Indeed, within the time frame of concurrent processes, an object can be both a server and requester. For example, a dbms module could act as a source of replies (server) when retrieving records in response to query requests and as a source of requests (requester) when requesting that these records be printed by a peripheral management module. A representation of this duality of object functions is shown in Figure 4b. An example of an implementation of this model on a local computer network, using a control bus for control messages and a data bus for data messages, is described in [Schneidewind 82] and illustrated in Figure 4c. It is interesting that some physical implementations of local networks utilize separate data and control busses [Kuhns 79]. It is important to note a key distinction between the use of links as implemented in DEMOS and ROSCOE and their use in our design. In the former, the links are established in advance of communication between two objects or tasks. This approach facilitates establishing a capability or legitimacy for sending and receiving and for reserving buffer space at O_j . In our design the links are created upon receipt and acceptance of the first message by O_j from O_i . Since our objective is to increase speed and reduce overhead, we feel that handshaking procedures should be cut to the bone. We accomplish this objective via the following procedures:

- ° O_i sends a message whenever it has one to send.
- ° O_j validates incoming messages as being legitimate for it to process. Invalid messages are handled with an error message to O_i and to the system administrator module (i.e., Recovery

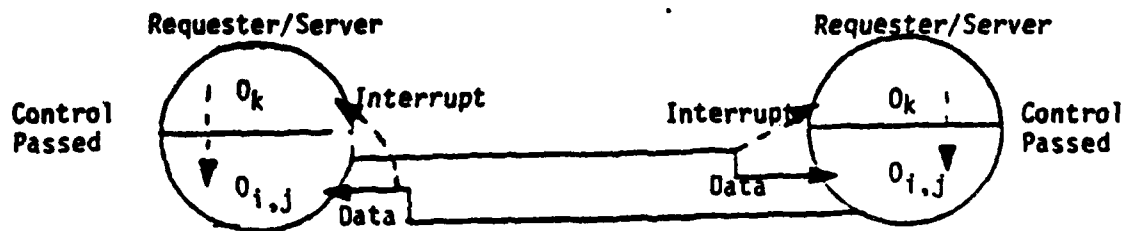
Management (RM) Module). An acknowledgement is sent from O_j to O_i when a received message is validated at O_j .

*A certain amount of memory is dedicated to input buffer space for O_j . If the buffer is full when a message arrives, O_j will reject it.

*A timer starts at O_i when a message is sent. If a time-out occurs, the message is re-transmitted. If two retries fail, an error message is sent to RM.

*Degree of asynchronism in communication: Fast communication can be achieved if a message can be sent using an interrupt (hardware interrupt for interprocessor or software interrupt for intra processor) to signal the fact that a message has arrived at a receiving object. This representation is shown in Figure 5a, where the operating system kernel object O_k assists the receiving object O_j by processing the interrupt, analyzing the priority of the incoming message and allocating the processor to O_j , if the message is the highest in priority of any ready processes, in a manner similar to that used in the Chorus Operating System [Guillemont 82]. Independent of this action, the message is stored in the dedicated buffer space of O_j . Another service provided by O_k is to replenish the buffer space of O_j from a buffer pool when the amount of space falls below a pre-determined threshold.

In contrast, the Tandem operating system link procedure is shown in Figure 5b [Bartlett 78]. The requester sends a message which is queued for the server. During the LISTEN phase the server checks for the presence of messages in the queue. If a message is present, the server will obtain a copy of the message during the READLINK phase. Next, the



O_k : Operating System Kernal

Figure 5a ... Interrupt Driven Object Communication Procedure

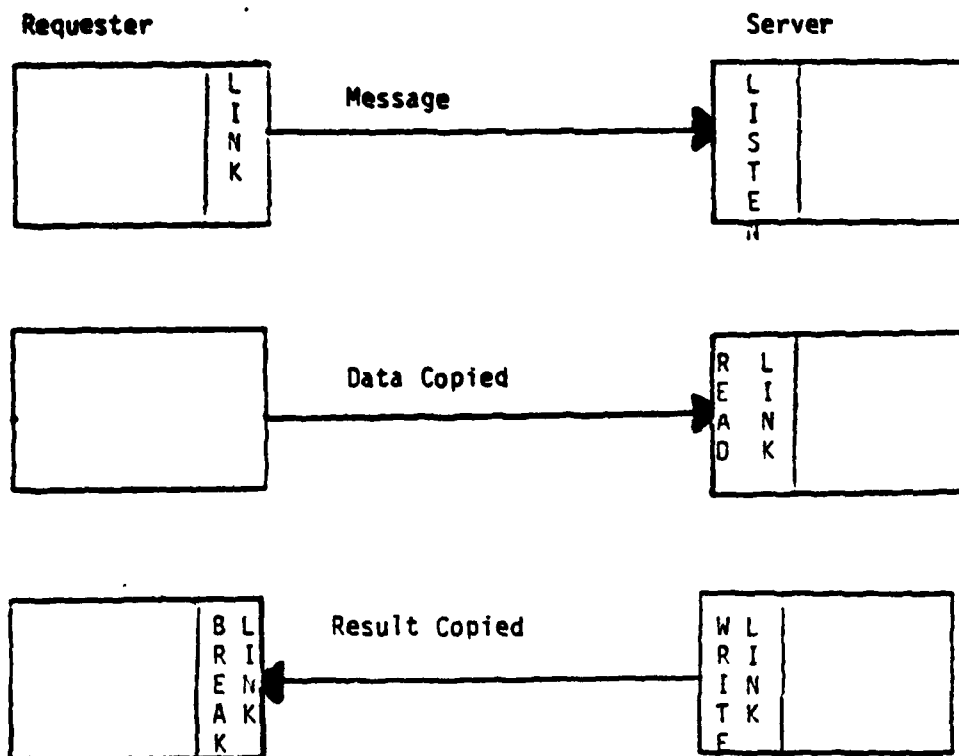


Figure 5b ... Tandem Operating System Link Procedure

server will return any result to the requester during the WRITELINK phase. Finally, the requester will terminate the transaction by calling BREAKLINK. The above procedure has the server calling LISTEN when it wishes to check for the existence of message, rather than being notified immediately, via an interrupt, of the presence of a high priority message, as in Figure 5a, thus impacting on the ability to provide real-time response.

Message Based Communication: this type of communication in a distributed system is advantageous for four reasons: (1) contention for the use of shared memory (the other major method of communicating in multi processor systems) is eliminated. However, it should be noted that we may simply be trading memory contention for bus contention, if a message-based system is used on a contention bus. (2) the critical section problem (i.e., prevention of damage of data in shared memory by multiple processes which share the data) [Ousterhout 80] is eliminated. (3) Operating with its own local memory for buffering messages, a process can be activated asynchronously to send or receive a message. This provides a high degree of concurrency because there is no need to provide process synchronization with semaphores or monitors for data protection purposes, as is the case when processes communicate via shared memory. The very act of message exchange between sender and receiver (i.e., send followed by acknowledgement) provides a natural form of process synchronization [Cheriton 79, Jones 79, Moore 82, Wood 82]. (4) The message communication sub-system is the only part of the system which must distinguish between local and remote (i.e., over a long distance network) object access [Donnelley 79].

Object Naming and Addressing: identifying (naming) and addressing objects in a large distributed system is a challenging task because:

(1) The user should not be burdened with remembering and providing object names and addresses to the system. Achieving this objective usually requires an elaborate system for mapping names to addresses. In the extreme, it may even be desirable to relieve the user of knowing the names of programs and files and to provide a data dictionary (actually the network services directory) in the system for obtaining the names of objects when the user provides subject key words or character strings. More will be said about our proposed data dictionary in a subsequent section. (2) For recovery and performance purposes, mobility of objects is highly desirable (i.e., programs and files should not be fixed in location or limited to affiliating with only certain nodes). This implies that binding of names to addresses should be deferred as long as possible (i.e., just before a message is sent) [Saltzer 82]. (3) Significant problems arise in attempting to maintain uniqueness of names in a large network, consisting of numerous local networks interconnected by a long distance network (e.g., SPLICE). It would be unreasonable to require or to expect that object names will not be duplicated among local networks. (4) From the standpoint of software maintenance, it is desirable to place all name-to-address mapping and routing information in one object (i.e., name server or data dictionary). On the other hand, from the standpoint of speed, the necessity to access the dictionary every time a name or address is

*Although not strictly correct, names and logical addresses will be synonymous in order to maintain consistency with previous reports.

required would be prohibitively slow in a network with a high message rate.

With regard to (1), it is extremely important for an object name to be the name of a service or module which provides the service, rather than the name of a node or network attachment point (e.g., in ARPANET, an IMP number and port number). If this is not the case, network services are location and hardware device dependent. Saltzer cites the undesirable situation in ARPANET of the name of a node or service, such as RADC-MULTICS, being in reality a network attachment point (IMP 18, port 0) associated with a Honeywell 68/80 host computer [Saltzer 82]. If this host were to be attached to the network through another IMP, either the routing tables would have to be changed to reflect this fact or the service would have to be given a new name.

The problem of binding, as mentioned in (2), can be simplified considerably by resorting to broadcast message transmission for intra local network communication. Many local networks (e.g., Ethernet.) provide a broadcast mode in which all nodes will recognize a transmitted message. In the broadcast mode a designated bit in the destination physical address of a message can be used to indicate the broadcast mode. When this mode is used, it is unnecessary for the sending object to obtain the address of the receiving object. All that is needed is the name of the receiving object, which can be obtained from a variety of sources: user, data dictionary or task table, which associates tasks with names (see later section). The data link layer in each receiving object would examine the name part of the destination logical address to determine whether the message is addressed to it, or use the kernel of the operating system for this purpose as shown in Figure 6. When a

message is a true broadcast type, that is the message is intended for all objects, the "Message Type" field will indicate this. As reported by Shoch, the original intent of the Distributed Computing System (DCS) - a ring network - at U.C. Irvine, was for objects to communicate by name [Shoch 78]. The success of this approach in the DCS design would have depended on hardware to recognize a broadcast message and an associative memory for comparing the name in the message with the names of objects resident in a node. This hardware was not implemented. A feasible alternative to achieving this capability is to use a software solution in a higher level layer (i.e., data link layer mentioned above) for determining whether a message is addressed to a particular object. A bus architecture is advantageous for this method of object addressing because its natural mode of communication is broadcast.

In case of a message which is destined for an object located in another network, this fact is indicated in the "Message Type" field. This type of message would be sent in the broadcast mode, like other messages, but only the network interface object (National Communication (NC) Module See Figure 6) would respond to it. The physical destination address would have been obtained previously from the data dictionary, with the help of Session Services, as shown in Figure 6.

The approach of using the broadcast mode of communication, coupled with using physical addresses only in the case of remote network communication, allows complete mobility of objects, including the NC Module.

A solution to (3) is to allow names and addresses to be duplicated within each local network*, but to concatenate the node address with the

* By this we mean that a name in local network A may be the same as a name in local network B, but two names may not be the same within A or B.

network address, so that the full address is unique across the entire network. Since a received message is examined first by physical address and second by name, or logical address, object names may be duplicated in local networks.

A compromise solution to the problem of (4) is to provide the names of frequently referenced objects (i.e., local network names) in the service table of each object; the remaining names (and physical addresses), which pertain primarily to remote objects, are obtained from the data dictionary.

Object Control

In the previous section our objects learned how to talk. If this communication is to be more than a babble among peer objects, some type of control structure must be imposed. The control mechanism is complex due to the following reasons:

- *A user process may have multiple sessions active at any time.
- *An FM can be active in multiple sessions at any time.
- *Two or more FMs can be active in a single session.
- *Message exchange between pairs of FMs can be nested. That is FM(A) may request a service from FM (B), which may, in turn, find it has to request a service from FM (C) in order to complete the service for FM (A). This results in a multi-tasking mode of operation.
- *Message exchange can involve remote (i.e., over the Defense Data Network) as well as local communication.
- *Some tasks are interactive, while others are completed on a deferred basis.

Although the design philosophy is that of distributed systems, the complexity of the processing environment requires that user terminal

processes be given considerable assistance in carrying out their tasks. This assistance is provided by the Session Services (SS) Module [Bachman 78]. User terminal processes specify task requirements, largely by task name, and with the assistance of the data dictionary, where necessary. It is the responsibility of SS to provide the additional information and control which are necessary in order to complete the task. This could include the following, depending on circumstances:

- °Names and addresses (as required) of the FMs which are necessary for processing a task.

- °Instructions or service codes for FM processing (to the extent that this is known in advance).

- °Invocation of the first FM (called the Controlling FM (CFM)), via a message which could also contain user data and authorization data, if another node is involved.

- °Receipt of a completion or error code from the CFM.

Figure 7 illustrates two instances of the control structure: one where three local FMs and one remote FM are involved and a second involving two remote FMs. Figure 8 shows this situation in greater detail, indicating two tasks, (1) and (2), and the invocation (FM calls) and return paths. Control can go from SS to a CFM, local or remote, thence to other FMs: local - local and local - remote but not remote - local. This restriction pertains only to FM calling sequences; data transfer between FMs is fully bidirectional. The degree of nesting (i.e., number of levels of FMs which should be allowed in a network) is a major research problem in distributed systems, which cannot be totally solved here. In concept, the total network could be considered as one large pool of objects which could be nested without limit, and in any

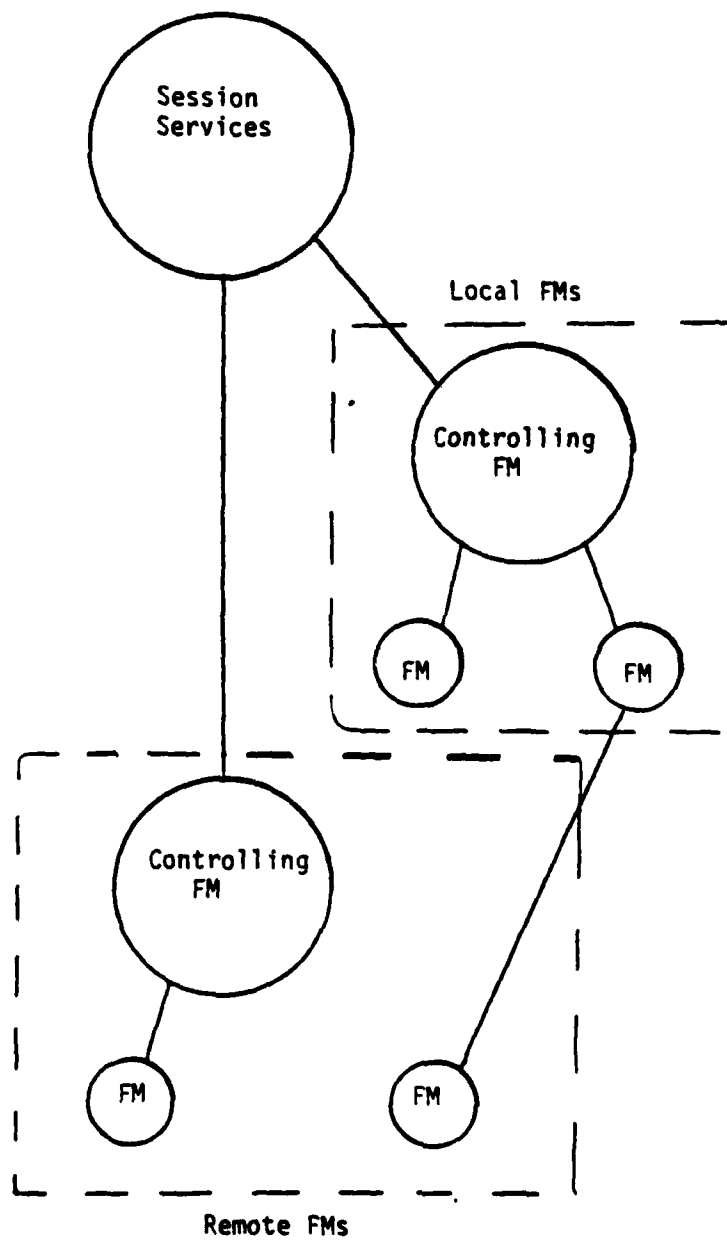


Figure 7 ... Object Control Structure (Two Instances)

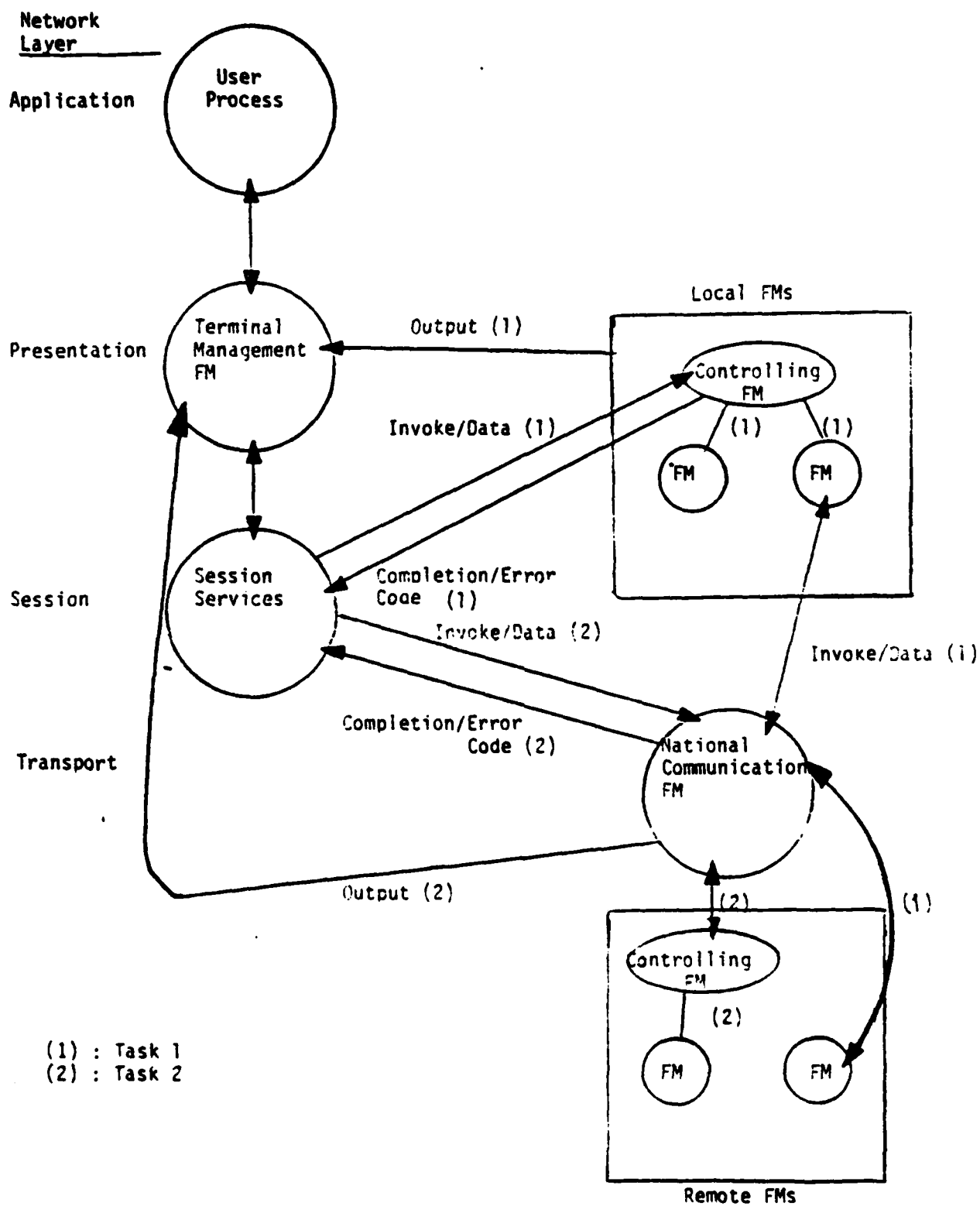


Figure 8 ... Object Invocation and Data Paths (Two Instances)

sequence, in order to accomplish a task. This unconstrained approach could pose serious problems relative to response time performance, security, data integrity and recovery.

PERFORMANCE EVALUATION

Now that our system has evolved to maturity, with objects identified, names and functions designated for objects, and a mechanism established for object communication, we need some simple metrics to evaluate the performance of our design. This will be only a brief discussion of possible metrics, which are relatively simple to apply. The development of analytical and simulation models for evaluating these systems is a complex mathematical process. A model of the local network will be covered in a future report. Our purpose now is limited to presenting several easy-to-apply metrics which we hope will assist in comparing various network topologies and protocols.

The performance metrics which appear below are defined in terms of communication among nodes (e.g., processors). This is the most meaningful interpretation of inter object communication in a performance context, since a message must first be addressed to a node - perhaps by broadcast transmission - before the message can be received by higher level objects (e.g., functional modules and processes). In addition, the metrics are restricted in applicability to intra local network communication, since inter local network performance is dependent on the topology and delay time characteristics of the interconnecting long distance network. The same set of metrics applies to all local networks source or destination, with the important qualification that destination local network performance metrics must be augmented by the metrics of

the long distance network (beyond the scope of this report) in order to provide realistic measures of performance.

Selected Local Network Metrics

The metrics which are defined below are compared for bus (carrier sense multiple access), token ring (unidirectional) and star topologies in Table 0, for an N node network.

1. **Accessibility:** The number of nodes which can be reached directly (i.e., without traversing intervening nodes) with a single message transmitted from a given node.
2. **Connectivity:** The number of nodes which a given node is connected to.
3. **Average Nodes Traversed:** The sum of the intervening nodes traversed by messages transmitted by a given node, in order to reach each of the other N-1 nodes, divided by N-1. This quantity is directly related to the delay time introduced by nodal processing.
4. **Average Message Delivery Time:** The average of the sum of the time to acquire the medium (e.g., bus) or obtain control (e.g., token) plus message transmission time.

TABLE 0

LOCAL NETWORK PERFORMANCE METRICS

	<u>CSMA BUS</u>	<u>TOKEN RING</u>	<u>STAR</u>
1. Accessibility	N	1	0 (Note 1)
2. Connectivity	N	2	1
3. Ave. Nodes Traversed	0	(N/2)-1	1 (Note 1)
4. Ave. Message Delivery Time	$T_a + T_t$ (Note 2)	$N(T_t + T_n) - 2T_n$ (Note 3)	$2T_t + T_n$ (Note 4)

Definitions

N: Number of nodes.

T_a: Average bus access time. This is a function of probability of acquiring the bus which, in turn, is a function of N [Metcalf 76].

T_t: Average transmission time on a bus or single link of ring or star = message size in bits/bits per second transmission rate. Signal propagation time is negligible and is ignored.

T_n: Average delay time incurred in a node of ring or star. Varies for ring depending upon whether the token is only passed through a node or whether the node captures it and then transmits a message [Tropper 81, Yuen 72]. For a star, there is only one instance of this delay time, as the message traverses the central switching node.

Notes

1. Assuming central node used as a switch for other nodes.
2. Bus access time + bus transmission time.
3. The time for the token to circulate one-half of the ring distance (assuming no intervening node captures token) plus the time for the message to be transmitted one half of the ring distance, minus two nodal delay times to account for transmitting node not being an intervening node in token capture and message transmission.
4. Transmission time on two links: source node to central node and central node to destination node plus delay time in switching node.

As can be seen from Table 0, the bus has very good properties with respect to a message reaching other nodes without incurring nodal delay time T_n . Furthermore, since T_t will be equal for all local networks with the same bandwidth, comparative performance between a bus and ring, for example, hinges on the relationship between T_a and T_n . Both of these variables are a function of N (i.e., a function of total offered load) and traffic rate. When the load is light, the probability of acquiring the bus is high, and T_a is low relative to the time for a node on a ring to acquire the token (function of traffic, N and T_n). At high load, the probability of acquiring the bus is low and bus acquisition time is unbounded. Under high load the token acquisition time increases, but this time is bounded because a given node is guaranteed to acquire the token in finite time by virtue of the sequential nature of token passing.

The problem of unbounded access time for a random access bus is solved by using a token passing bus, but at the price of incurring higher overhead for all transmissions.

DISTRIBUTED SYSTEM DESIGN STRATEGY

Now that the paradigm has been used to: (1) explain how distributed systems evolve, (2) identify their general properties and (3) describe specific characteristics which are necessary for efficient distributed system operation, we are prepared to delineate a design strategy for achieving these characteristics. In distributed systems and local computer network architectures, the characteristics of objects and the manner in which these objects communicate are very important considerations in the design of these systems. Since a distributed

system does not rely on centralized control, much of the burden of providing effective coordination of diverse objects falls on the communication subsystem and the method of interprocess communication. In order to provide reasonable response time to the user and to avoid excessive resource consumption attributable to support functions, the communication and interprocess subsystems should be as simple as possible, consistent with the need to maintain state information for recovery purposes.

With this general guideline in mind, various design principles are listed below which are applied to the design of a distributed operating system for SPLICE (described in a later section).

FUNCTIONAL MODULES

*A minimum amount of state information (e.g., status of transaction processing) should be held by each FM. If this principle is adhered to, it will not be necessary to reconstruct transaction status information for the numerous FMs which could be involved in processing multiple concurrent transactions. A better alternative to maintaining extensive state information in individual FMs is to centralize this information in the Session Services (SS) Module, the module which has responsibility for monitoring user sessions. The centralization of recovery procedures is an exception to the general rule of dispersing responsibility for functions across modules in a distributed system. The primary reason for this exception is that recovery requires a global view of system status. A global view is not available from individual FMs; it is available from Session Services. Also, changes in recovery logic would be confined to changing SS and the recovery management (RM) module by using this approach.

*Contrary to the foregoing principle concerning the preservation of recovery information, as much processing intelligence as possible should be placed in each FM, thus limiting dependence on intermediary FMs and reducing the associated message traffic, consistent with requirements for modularity, recovery and transaction back-out. This policy is followed in order to localize decision making and to make it unnecessary for the FMs to have frequent need for the services of an operating system executive, thus reducing overhead delays, at the price of some duplication of code in the FMs. In order to meet these conflicting requirements (e.g., modularity vs. speed), it will be necessary to decompose FMs into submodules.

*Information which is required by an FM to understand which task to perform when a message is received and where to send the results after the task is completed, is contained in a service table accessible by the FM. An FM finds out what to do by processing service codes, which have been placed in the message by SS, against the service table and passes the result to the 'calling' FM or to the next FM in the processing sequence, depending upon circumstances. In addition, the table tells the FM which transactions are legitimate for it to process and where to report error conditions. This procedure simplifies software maintenance by confining changes in transaction processing logic to changing the service tables and avoiding changes to FM code.

*Session Services coordinates FM activity and provides, to the extent possible, work instructions via the service codes it inserts in messages to the FMs. In some cases, work breakdown cannot be completely determined in advance by SS because the sequence of operations may be data dependent or highly interactive. In such cases, SS merely passes

control to the first FM which is to perform an operation and subsequent 'calls' to other FMs, if any, take place according to processing conditions. Session Services retains and maintains state information until either a completion message or error message has been received from the controlling FM.

Characteristics of Message Exchange Between the Sending FM [FM(S)] and the Receiving FM [FM(R)].

°When FM(S) has a message to send, it sends it without establishing a link with FM(R) beforehand or asking permission of FM(R) to send a message.

°FM(R) does not wait for a message; rather, it reacts to a transmitted message by means of a call from the operating system kernel. It does not listen for a message nor does it need to grant permission to FM(S) to send a message.

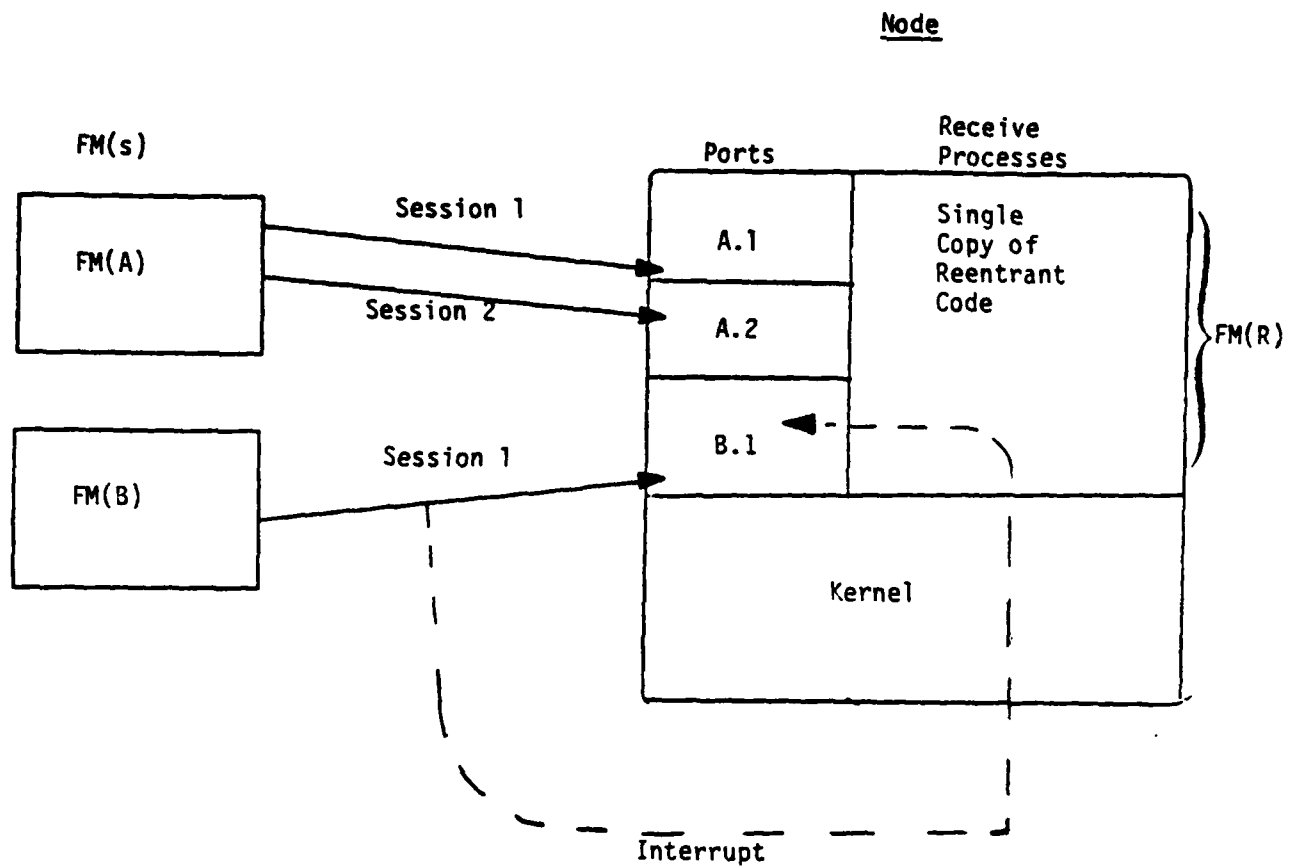
°No exchange of messages (handshaking) takes place prior to transmission. There is no need for this because: (1) If FM(R) is down, FM(S) will time out and repeat the transmission until it receives an acknowledgement from FM(R) or gives up and reports an error condition to the Recovery Management (RM) Module or (2) FM(R) will ascertain, via table look-up, whether this is a valid message for it to process. If not, it will send a negative acknowledgement to FM(S) and also report it to RM.

Operating System Kernel

°The kernel of the operating system should perform a minimum number of tasks [Boebert 78] and only those which are common to service all FMs residing in a processor (e.g., memory management). This is done to minimize redundant FM code. A copy of the kernel would be resident in

each node. Any function which is unique to an FM should be provided by it, rather than by the kernel. The kernel acts on interrupts, as shown in Figure 9, and allocates an FM(R) receive process to the appropriate port. In Figure 9, a concatenation of FM(S) name and session number are used as a port address. Each FM has a list of valid FMs with which it is allowed to communicate, thus providing FM(R) a name to match to the name in the received message, thus contributing to system security. The session number is assigned by SS. In addition a sequence number, not shown in Figure 9, is assigned by SS in order to ensure correct message sequencing for internetwork traffic. Port buffer space is dedicated to messages received by FMs with high input rates and dynamically allocated for FMs with low input rates. The minimum port buffer space is equal to the maximum message fragment size of a single message. A single copy of reentrant code provides multiple receive processes, as shown in Figure 9. This feature, combined with the dedicated ports, allows a high degree of message multiplexing through an FM(R).

The mechanism of Figure 9 illustrates the message oriented approach in which no link is established in advance of communication between FM(S) and FM(R) nor is agreement concerning port addresses required before a pair of objects can communicate [Akkoyunlu 74]. With this approach, a message is addressed to a process and implicitly to a port by virtue of the name of the FM(S) and its associated session. The connection - oriented approach, on the other hand, has ports which are independent of particular processes. The former approach is used in this design because of its lower overhead, consistent with the argument of an earlier section about the need to reduce communication set-up time. It should be noted that, in our design, a port is not a logical



FM(S) : Sending Functional Module
 FM(R) : Receiving Functional Module
 Port Address : X. Y, where X is name of FM(S) and Y is Session Number

Figure 9 ... Message Assignment to Ports and Use of Kernel to Process Interrupts

data path for sending a message from one process to another, as in [Walden 72], but rather an identification of a storage bin at FM(R) for storing and sequencing the messages from a given FM(S) and session.

The kernel supports interprocess communication with hardware interrupts (interprocessor communication) and with software interrupts (intraprocessor communication), as suggested in Figure 9.

Network Operating System Design

After having laid the theoretical foundation for the design of a distributed network operating system, we turn now to the details of implementing such a system. The emphasis in this section is on the design of a local area network operating system (LANOS) for supporting intra LAN communication and logistics transaction processing. Functional design specifications for inter LAN communication (over the Defense Data Network) are described in [Schneidewind 82]. The operating system design for inter LAN communication will be provided in a future report.

This section is organized as follows:

First, Specific Operating System Design Objectives are described, in order to key the discussion to the details of the operating system design which follows.

The following topics are then covered:

- Interprocess Communication
- Scheduling
- Deadlock Prevention
- Memory Management
- Shared Resources and the Critical Section Problem

Next, the User Interface Specifications, or user view of the system, is presented, consisting of the following:

- User Calls

- Dictionary/Directory System

Last, the computer hardware environment, in which the operating system will operate, is given in the Computer Architecture part.

An overview of the layered operating system design is shown in Figure 10.

Operating System Design Objectives

The objectives of the design approach are speed and low overhead. One implication of this objective is that there will not be a central supervisor. Each processor, and functional modules which it serves, will have its own operating system services, which will be largely implemented in micro code for speed advantage purposes. Replication of function and greater use of memory is the price incurred in order to reduce overhead and increase speed.

The traditional operating system approach is for one copy of the operating system to be available to all user processes and for the user processes to invoke the operating system via a trap or supervisor call instruction. The processor performs a context switch to that of the supervisor, executes the function and returns to the user process. In a bus-oriented multiprocessor system, the processors which do not contain the single copy of the operating system execute much slower than the one processor that does contain the copy of the operating system in its local storage. This occurs because of bus contention incurred by processors which must make remote memory accesses to the operating

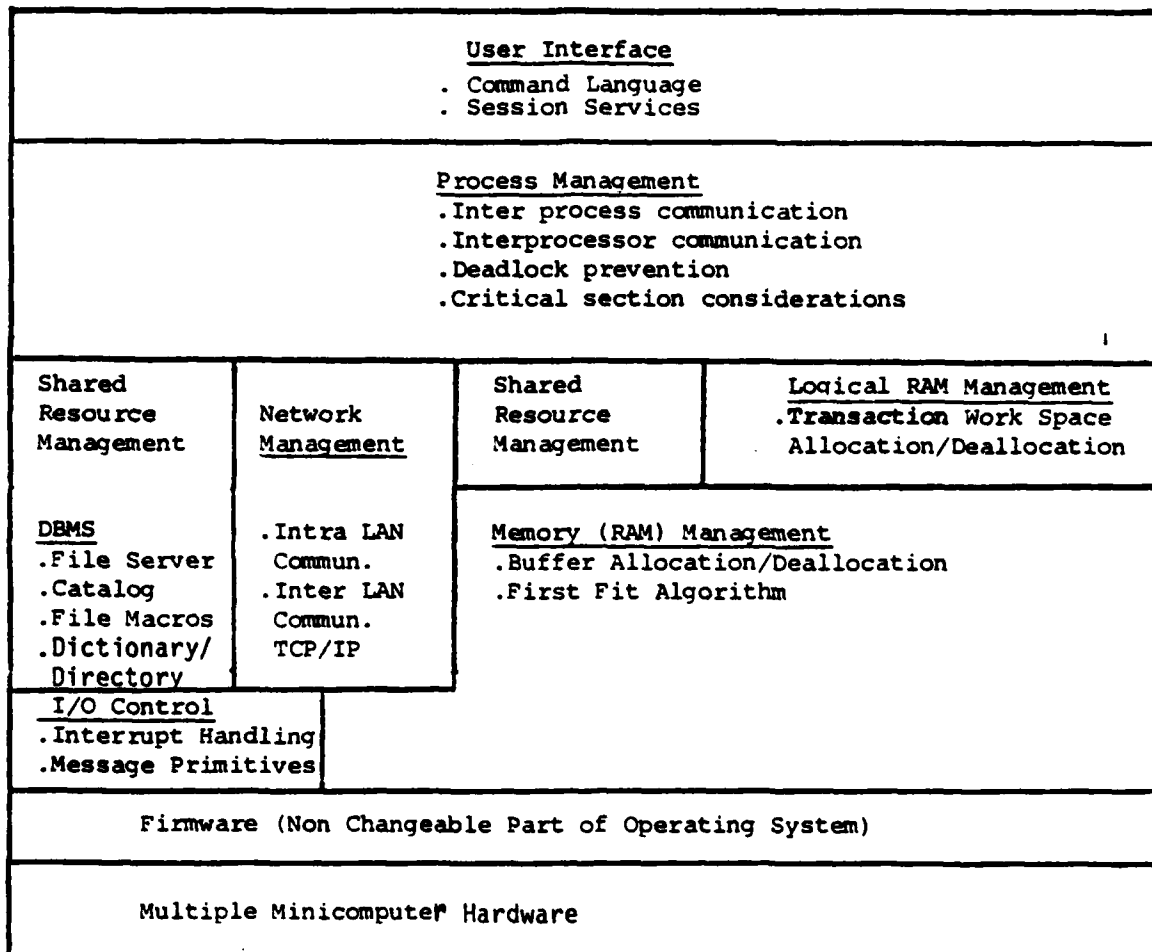


Figure 10 . . . Layered Operating System Design

system code, via the bus. One way to prevent performance degradation is to provide a copy of the entire operating system in every processor's local memory, but this may be infeasible due to the size of the operating system and required memory size. The solution used in the Medusa distributed operating system is to distribute the various operating system functions (i.e., process management, memory management and file management) to different processors. [Ousterhout 80]. Each processor is dedicated to performing a single function. Operating system functions are invoked by messages containing parameters which specify the desired service.

In the LAN operating system, functions are associated with each processor and serve the functional modules which are resident in a processor. In the extreme case, in the LANOS, there could be a single FM per processor. This would simplify interprocess and interprocessor communication because it would allow a single consistent interprocess signalling method to be utilized. All communication between FMs would be accomplished by putting a message on the bus (broadcast mode). Each processor would read the message and copy it if it is addressed to it; otherwise, the message is ignored. From the adaptor, the message would proceed via DMA to the processor. Unfortunately, putting a single FM in each processor would require a minimum of eight processors. The bandwidth required for the minicomputers contemplated for the LAN could not be easily accommodated with existing commercially available LANs. More important, there would be excessive bus contention involved (i.e., every time a message had to be transmitted between FMs, it would have to be placed on the bus). Therefore, FMs which have a great need to communicate are grouped in the same minicomputer.

This will result in both interprocessor and intraprocessor signalling, but bus traffic will be reduced significantly. Also, greater uniformity in minicomputer capacity (i.e., RAM size and disk size) will be achieved. This is important from a reliability and recovery standpoint because FM size varies considerably - from a complex data management module to a relatively simple peripheral management module. If there were one FM per processor, either there would be enormous variation in the capacity of individual processors, or there would be significant wastage of processor capacity, if processors were of equal capacity. In the former case, a failure in a large processor (where the DBMs is resident) would not allow a smaller processor to take over the functions of the large processor in a degraded mode. Related to this point is the desirability of providing mobility of FMs and physical location independence so that, depending upon the availability of hardware, FMs can be moved from one processor to another. Furthermore, hardware maintenance and the possibility of future hardware upgrades is simplified by providing processors with similar, although not necessarily identical, capacities. The nature of the interprocess and interprocessor message communication system, which is used to implement this concept, is described in the next section.

INTERPROCESS COMMUNICATION

1. Definitions

A. Functional Module

A functional module (FM) is a unit of software which performs a single major function such as terminal management, data base management, peripheral management, etc.

B. Process

A process is the execution of a functional module.

C. Interprocess Communication

The processes of two functional modules communicate when a message is transmitted from the transmitting functional module FM(T) to the receiving functional module FM(R).

D. Message

A message may be a user initiated transaction or a system generated control message (e.g., acknowledgement).

2. Method of Interprocess Communication

As indicated earlier for interprocessor communication, this occurs by the FM(T) transmitting a message on the bus, all FM(R)s reading the message, and the addressed FM(R) copying the message into dedicated input buffer space which is reserved for the given FM(T). A message is transmitted when an FM(T) has a message to transmit, as a result of completing a processing step, and without the approval of "higher authority." For intraprocessor communication, an SVC type of interrupt is generated, the executive is called and concurrently the message is stored in a dedicated input buffer space which is reserved for the given FM(T). Every correctly received message is acknowledged by FM(R) to FM(T) and only one message can be unacknowledged between a pair of communicating FMs at any instant in time. A timer is set into operation by FM(T) when the message is sent. A message will be retransmitted if the timer interval expires prior to the receipt of an acknowledgement from FM(R). If this happens three times, further communication between FM(T) and FM(R)

ceases and FM(T) reports the situation to the Recovery Management (RM) module for corrective action. If FM(R) should receive a message when its input buffer is full, it will discard the message. In addition, FM(R) will have a duplicate message check. Because of the stop and wait nature of the transmission, FM(R) will only have to hold one message.

3. Program Control Block (PCB)

The PCB [Deitel 82] will be maintained by the Executive in each processor. It will contain the following information concerning each process:

°Identification

-Based on maximum number of processes allowed

°State (Ready, Blocked, Running)

°Reason for being Blocked (Put to Sleep)

-I/O Wait

-Waiting for message acknowledgement

-Resources unavailable (lack of memory, etc.)

-Etc.

°Current number of processes

°Maximum number of processes allowed

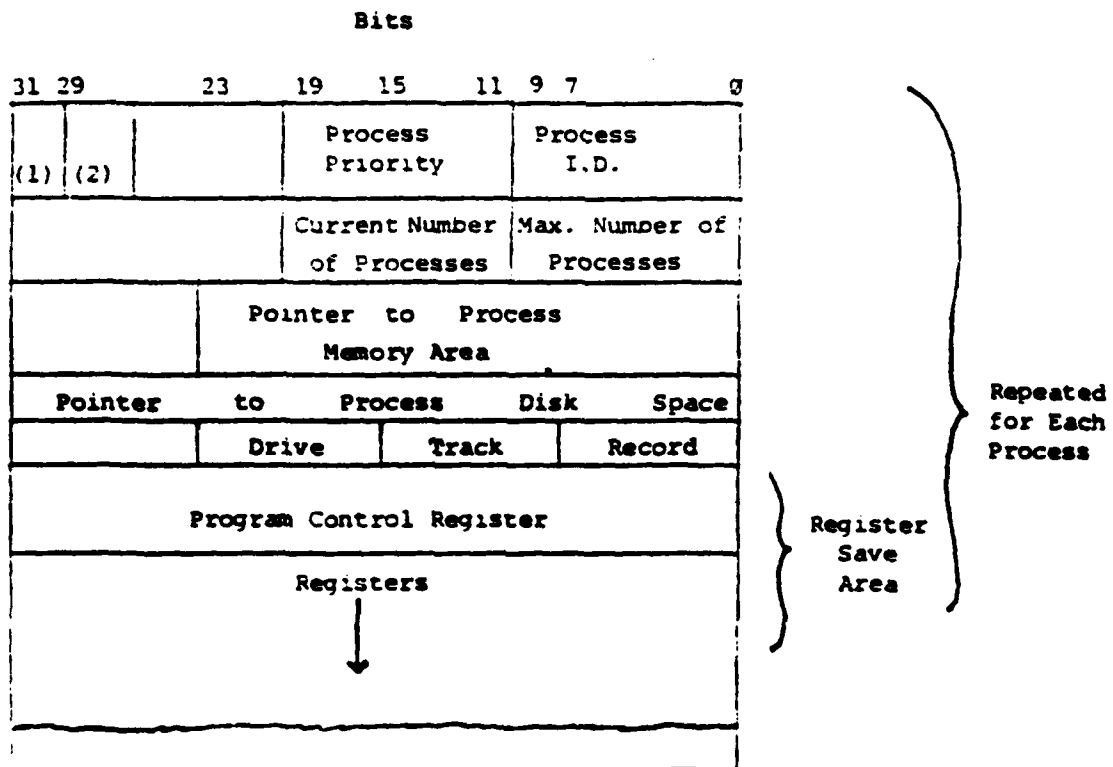
°Register save area

°Pointers to resources being utilized (memory, I/O, disk space, etc.)

The layout of the PCB is shown in Figure 11.

SCHEDULING

The LANOS is interrupt driven: a process continues to execute unless interrupted. The highest priority ready process is always executed after recognizing the interrupt. There is no time quantum



Notes

- . Process ID, Process Priority, Current Number of Processes, Max. Number of Processes: 100 users max., 10 Processes per user max. → 10 bits.
- . (1): Process state: Ready, Blocked, Running.
- . (2): Reason for process being blocked: I/O Wait, etc.
- . Pointer to Process Memory Area: Physical Address Space limited to 16 M Bytes → 24 bits.

Figure 11 ... Program Control Block Table

allocated to each process nor are processes executed in a round robin fashion. The reason for this is that the system is designed to provide transaction processing, not a general purpose, interactive, time sharing service. This system will process various transactions at different priority levels. The priority is provided by the user at the terminal, when the transaction is entered, or is inherent to the type of transaction. The use of the processor for initiating I/O and recognizing and processing interrupts will take precedence over computational processing, because the application processes are I/O oriented. Appearing below is the process priority, going from highest to lowest priority.

°Operating system processes

- System malfunction and recovery events
- All other operating system events on FIFO basis

°Application processes by transaction priority

°Within a priority class:

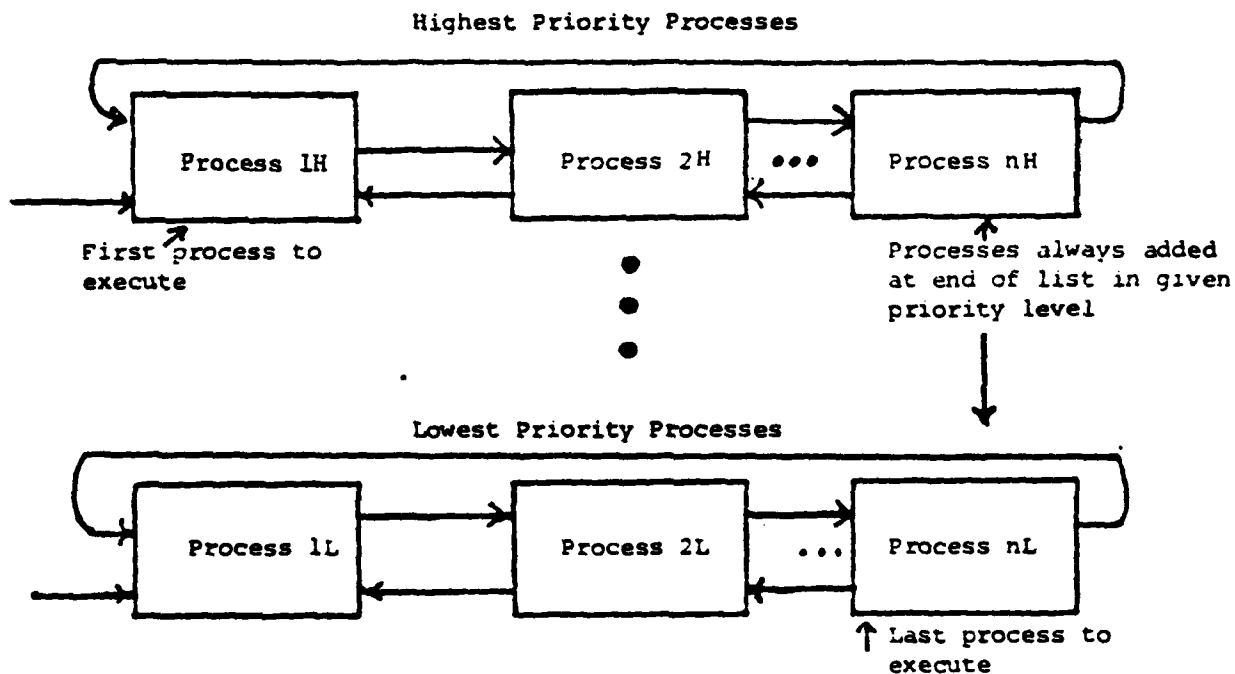
- Output from a FM
- Input to a FM
- Processing for a FM

Although a time quantum is not utilized, a process is not allowed to execute indefinitely. When a process has executed for the maximum allowable processor time, a timer interrupt will cause that process to be put in the ready queue at its original priority level. The data structure for the ready list is shown in Figure 12.

DEADLOCK PREVENTION

The four necessary conditions for a deadlock to occur are the following (Coffman 71):

Linked lists (Queues - FIFO)
of processes of equal priority



A process is deleted from its list when it blocks and is reinstated on its list (at the end) when it unblocks.

Figure 12... Ready List Data Structure

- (1) Processes claim exclusive control of the resources they require (mutual exclusion condition).
- (2) Processes hold resources already allocated to them while waiting for additional resources (wait for condition).
- (3) Resources cannot be removed from the processes holding them until the resources are used to completion (no preemption condition).
- (4) A circular chain of processes exists in which each process holds one or more resources that are requested by the next process in the chain (circular wait condition).

Only one of the above conditions must be violated in order to prevent deadlock. The method chosen is to violate (2) by dedicating all the space (memory and disk) to a FM and each of its submodules which would be required to process a message from each of the FM (Ts) with which it is currently communicating. In addition, there can only be one unacknowledged message existing between a given FM(T) - FM(R) pair at any time. FM (T) will not transmit another message to FM(R) until the previous message has been acknowledged. A FM(R) will not need to request more space. It will have it pre-allocated (see Figure 13). If a message arrives due to timeout at FM(T), failure in acknowledgement, or arrives before the buffer is cleared, the message is discarded.

Finally, (3) will be violated by preempting a process from further use of the processor, if the amount of processor time consumed equals the maximum allowable time. This will also prevent the indefinite postponement problem (Deital 82).

MEMORY (RAM) MANAGEMENT

In keeping with the design objectives of speed and low overhead, memory management will be simplified as a result of doing the following:

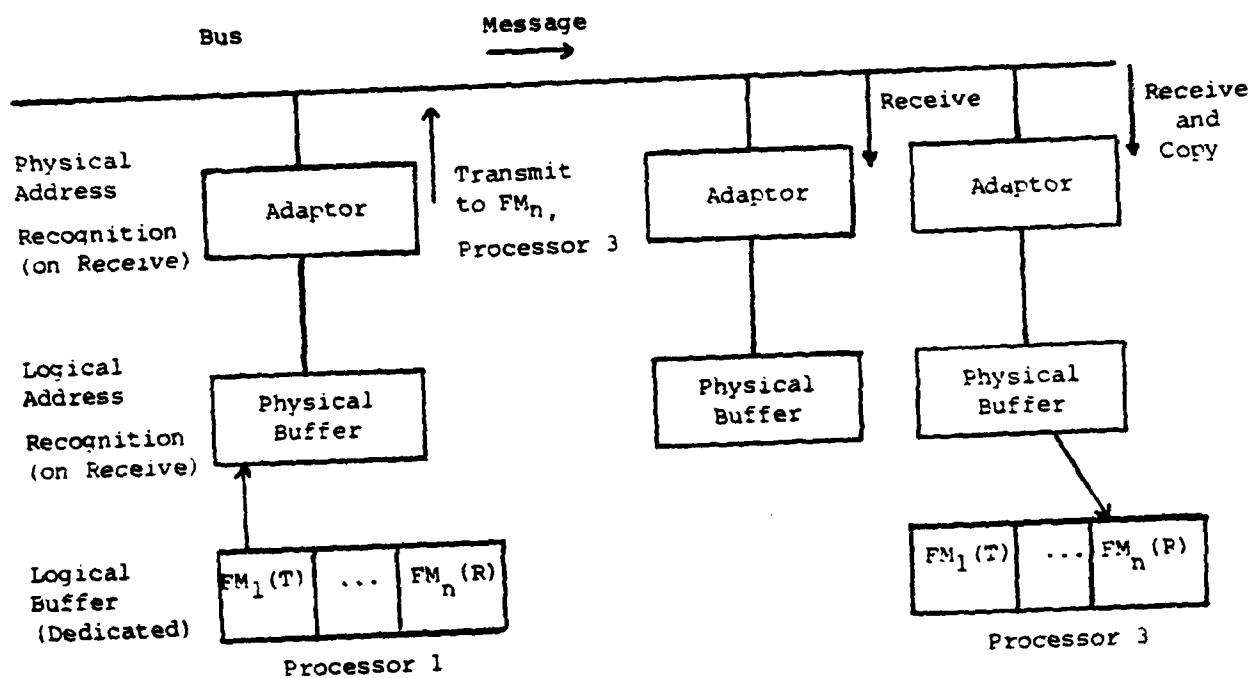


Figure 13 ... Buffer Allocation

- °Input and output buffer space will be pre-allocated and dedicated to each of the FMs with which a given FM can communicate (see Figure 14).
- °FMs are permanently stored in memory, but not in fixed locations (i.e., FMs can be relocated, depending upon the configuration which is available and the need to relocate modules as a result of recovery actions).
- °Virtual memory or swapping are not utilized. Therefore, FMs are not subject to paging or swapping.
- °The only dynamically allocated memory is the work space used by a FM. The amount of workspace which is needed is determined by user demands and is highly variable. Each FM will be assigned its own workspace, as needed, in order to speed up execution and to avoid the use of shared data, which would result in a critical section problem.

In addition to speed and overhead objectives, the above procedure is used because the use of the FMs is highly predictable: at least two FMs are used to service every user command. The frequency of use of all FMs is high. Therefore, if dedicated RAM (and disk) were not used, considerable time would be lost in continually allocating this space dynamically. The memory map is shown in Figure 14.

In order to conserve RAM space, and because all users share FMs via the use of a terminal command language, FMs are reentrant. A memory map table, using a first fit algorithm, and base and limit registers [Calingaert 82], is shown in Figure 15.

SHARED RESOURCES AND THE CRITICAL SECTION PROBLEM

The critical section problem arises when two processes attempt to access global variables simultaneously, thus destroying the integrity of these variables (i.e., a given process cannot be assured as to the

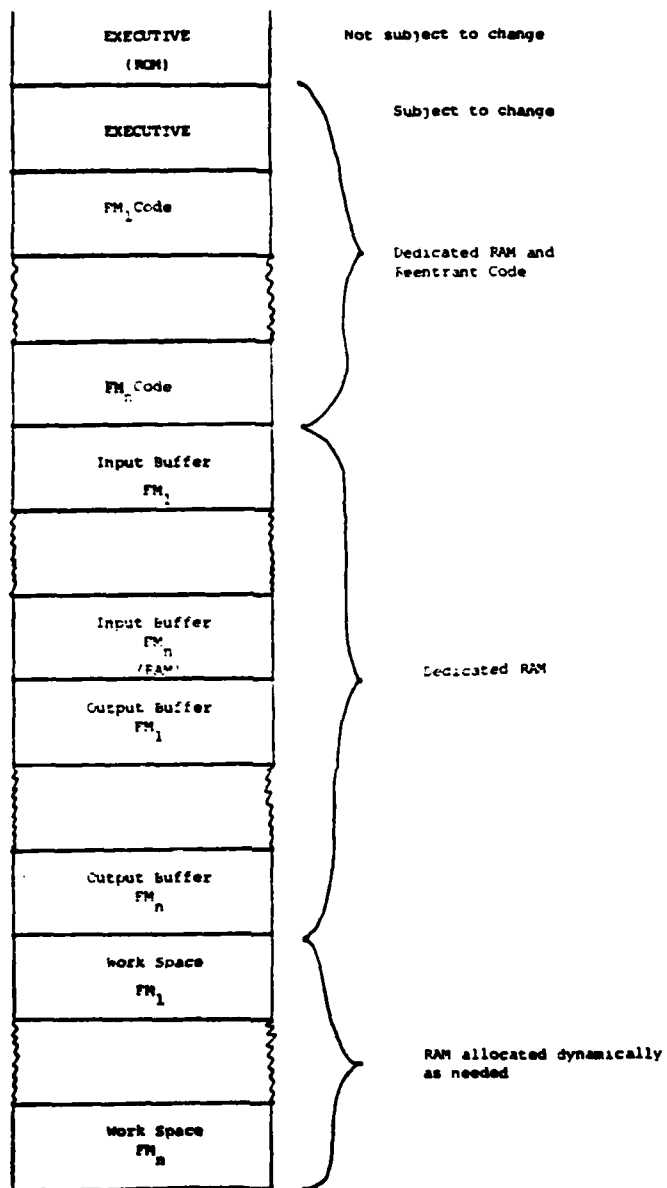


Figure 14 ... Memory Map

Number of Bits				
	9	3	1	19
	Process I. D.	Access Rights (R.W.E.)	Used/ Available	Address
0	X		1	0
1			0	4K
2			0	8K
3			0	12K
4	Y		1	16K
5	Y		1	20K
6	Y		1	24K
7	Y		1	28K
				32K
127				508K

Newly Allocated Work Space (First Fit)

Base Register

Limit Register

Notes

1. One entry for each 4K block in bank of 512K bytes.
2. Total of 128 entries.
3. Used = 1, Available = 0.
4. First fit algorithm used, in blocks of 4K bytes.
5. Beginning and ending addresses placed in Base Register and Limit Registers, respectively, when process is dispatched.

Figure 15 ... Memory Map Table

values of the variables at any instant in time). The portion of the code of a process which accesses the global variables is called its critical section. We must ensure that mutual exclusion is guaranteed. That is, the two processes must not be allowed to execute or enter their critical sections at the same time.

The one place in this system where resources are shared is described below. The solution to the critical section problem is to employ a variant of the monitor concept [Calingaert 82]. A monitor prevents a process from requesting a resource when that resource is being used (i.e., it ensures mutual exclusion). It also provides a mechanism for the process to release resources and return them to the resource pool. The implementation of the monitor is by means of the Resource Allocation module, as described below.

FMs bid directly for the use of additional reusable resources (e.g., memory, fixed head disks) which may be necessary for providing greater work space in order to conduct multi-tasking. A small Resources Allocation (RA) module assists the FMs in obtaining sharable resources. This module is associated with the Resources Status Table (RST). The RA, which is implemented in a dedicated processor, performs the function of shared resource allocation. It has available to it memory and fixed disk units which it allocates to the FMs on a shared basis. Naturally, FM processing which involves the use of shared resources will be slower per transaction than processing which uses dedicated resources because of transmission delays on the LAN and contention for shared resources. However, overall throughput would be increased by utilizing shared resources.

A module only bids for the use of shared resources, when it is presented with multiple tasks to perform and is unable to process them concurrently without the use of additional (shared) resources. The FM sends a resource request to RA, via a control message on the virtual "control bus", giving it the type and quantity of resource desired. In some cases multiple resources will be required. The RA module sends a "grant" message to the FM if the resource is available in the quantity desired; RA will then subtract the acquired amount of resource from the available quantity of resource in the RST. Upon receiving a "grant" message, FM will set an interval timer, via its executive, to a system-specified maximum value. Upon the expiration of this interval, the FM returns the resources to the pool by sending a "release" message to RA. The RA module then adds the released resource to the quantity available in the RST. The timer interval length can vary among FMs, depending upon processing priorities, and can be set by the System Operator. An FM must bid again, if resources are required subsequent to the release of resources. If the resource is not available in the desired quantity, the RA sends a "denied" message to the FM, which will continue to process with the use of dedicated local resources only. No record is kept by RA of this bid, and the FM must rebid at a later time, which is determined by an interrupt generated by a system-specified and operator adjustable timer interval set by the FM.

All of the above descriptions pertain to the use of control messages flowing on the "virtual" control bus. Once an FM has acquired a resource, it will send data (file records in some cases) to be stored in the resource unit and read data from the resource unit, with the assistance of RA. Once assigned by the RA, and until released by the

RA, the resources which have been granted to a FM are the "property" of the FM and cannot be shared by another FM. Data messages will be transferred on the "virtual data bus" and will be addressed to RA according to a two level address procedure (i.e., by the node in which RA resides and by the name of the RA module). RA must map between the message identification, as stored in a message by the FM, and the physical shared storage space. Upon receipt of a control message from the FM, requesting data from the resource unit and giving the message identification, RA will map to the physical storage locations, retrieve the data and send it to the FM.

Although the above "contention system" of allocating shared resources is crude in that resource requests are not queued, and requests will not necessarily be served on a FIFO basis, it has the great advantage of simplicity and low cost, due to the self-regulatory nature of the scheme. As soon as recordkeeping and queue maintenance are introduced to keep track of multiple requests - order of receipt, type and amount of request - the complexity rises rapidly. It is possible that sufficient dedicated local resources could be economically provided to each FM, such that an overload would rarely occur, and shared resources could be dispensed with entirely.

USER INTERFACE SPECIFICATIONS

User Calls

The following user commands, which will be expressed via a terminal command language, and implemented in the Terminal Management (TM) module, will be provided:

- *Read and display N file records.
- *Write N file records.

- *Read and print N file records.
- *Write and print N file records.
- *Copy N records.
- *Etc.
- *Invoke editor.
- *Invoke application module.
- *Invoke utilities and library routines.
- *Etc.
- *Change a file record in specified fields.
- *Delete N file records.
- *Insert a file record.

The above user calls will require the use of system primitives such as the following:

- | | |
|----------------------|------------------------|
| *SEND MESSAGE | *DEALLOCATE BUFFER* |
| *SET TIMER | *ALLOCATE WORK SPACE |
| *RECEIVE MESSAGE | *DEALLOCATE WORK SPACE |
| *ACKNOWLEDGE MESSAGE | *REQUEST RESOURCES |
| *ALLOCATE BUFFER* | *RELEASE RESOURCES |
| | *ETC. |

*Only used when more than the normal dedicated buffer space is required. Appropriate user and system parameters would be used with the above calls and primitives.

JUSTIFICATION FOR SPLICE DDS

SPLICE is an intricate, highly complex distributed network requiring the coordination and management of a wide range of information resources. The applications environment alone consists of major systems like IDA (Integrated Disbursement and Accounting), APADE (Automated Procurement

and Date Entry), UADPS-SP (Uniform Automated Data Processing System-Stock Points), and Trident LDS (Logistics Data System), each with its own set of data elements, files, programs, transactions, users, and reports. The need to manage these resources is critical, and in a distributed system even more so since they will be dispersed geographically and therefore more difficult to control.

One vehicle which can contribute significantly to this information resource management is a dictionary/directory system (DDS, variously known as "data dictionary", "data dictionary/directory", and "meta-data base"). A DDS is a set of one or more databases containing data about an organization's information resources which can be retrieved and analyzed using standard database management system (DBMS) capabilities (e.g.: query languages and processors).

The DDS has seen widespread but inconsistent usage in centralized processing environments [Curtice 81]. The primary advantages of a well-designed DDS which are applicable to SPLICE are [Allen et al. 82]:

1. a DDS provides a documented inventory of information resources;
2. a DDS provides a control mechanism for the analysis and design of new information resources;
3. a DDS provides resource independence.

The documentation features of a DDS would be particularly valuable for SPLICE. Referring again to the applications environment, a DDS would require users and analysts to define system data elements, files, etc. which would entail updating old definitions, discarding outdated ones, and introducing new ones. It would, in general, provide an

opportunity for establishing standards of data definition and description for application programs over the entire SPLICE system.

Once the DDS has been designed and implemented, it could serve as the focal point for further application program analysis and design. In particular, it could facilitate the conversion of a file-oriented application system to a DBMS-oriented system when DBMSs eventually become available on SPLICE. It could also assist in developing brand new programs by cataloguing data requirements resulting from requirements analysis activities [Teichroew and Hershey 77].

The ultimate power of a DDS resides in the resource independence which it provides, that is, resources are protected from changes in other resources (e.g.: data entities can be modified without modifying the application programs which access them and vice versa). This allows a high degree of resource usage flexibility which is especially vital in a distributed environment. A SPLICE DDS which enforced uniform resource description standards, for example, would enable large application systems like APADE and UADPS-SP to interface much more smoothly and conveniently.

The potential benefits of a DDS for SPLICE are significant but caution must be exercised. Few standards exist for DDS usage within centralized environments [Curtice 81] and fewer still for distributed environments. The following sections examine some of the issues which must be addressed in the design of a DDS for SPLICE.

SCOPE OF DDS

Dictionary directory systems have traditionally been viewed as data-oriented and either tightly or loosely coupled with DBMSs. Only recently has the role of a DDS been viewed in more comprehensive terms

as an information resource management tool [Allen et al. 82]. In this context, data is only one of the resources a DDS keeps track of. Other resources might include programs, files, hardware (concentrators, multiplexors, CPUs, etc.), user accounts, and reports.

This resource-oriented approach is strongly recommended for the SPLICE DDS over the more restrictive data-oriented concept as typified by the COBOL DATA DIVISION or existing data description languages. SPLICE consists of many diverse elements, all of which need to be managed but only some of which are data elements. This is explicitly recognized in the SPLICE specifications which call for a configuration management system (CMS) which will store and retrieve information on hardware, software, and documents pertaining to SPLICE configuration [SPLICE 81]. Clearly, the CMS is meant to serve similar functions to a DDS for these particular resources. Incidentally, this suggests that the SPLICE DDS might be built from the CMS, or alternatively, that the CMS might be subsumed by the DDS.

Although a resource-oriented DDS is a generalized and flexible tool, it raises the knotty question of "where does it fit within the SPLICE network?" If the DDS were strictly data-oriented, it would be easy to respond that it should be part of the data management module. Because it contains information on where different resources are located, however, and because several functional modules may use this information to provide message routing or other vital system resource allocation tasks, it can be persuasively argued that the DDS should transcend data management and be an integral part of the network operating system itself. Independent of the DDS's relation to the operating system, however, the data management module must still be

invoked in order to access the DDS since this involves database access. As a result, even though the DDS is more broadly associated with resources, it is still reasonable to make it part of the data management module even at the cost of possible information redundancy or access inefficiency.

CONTENTS AND LOGICAL STRUCTURE OF DDS

Since the DDS is resource-directed, it will contain more than just information about data. In particular, the following resources should be represented: data, hardware, software, transactions, personnel, and documents. Entities, attributes, and relationships associated with these resources are presented below.

Data resource information should include the following entities: data elements, data groups, schemas/subschemas, records, files, and databases. Attributes for these entities must be determined by the anticipated usage of the DDS. Typical attributes for the data element entity are suggested by Allen et al [1982] in Table 1. These attributes have been culled from existing commercial data dictionary/directory systems. Attributes for the file entity have been suggested in the SPLICE specifications [SPLICE 81] and appear in Table 2.

TABLE 1

Data Element Attributes (from Allen et al [1982])

Type	Language names
Range	Repetitions
Length	88 Levels
Unit of Measure	Key
Usage	Default value
	Display format

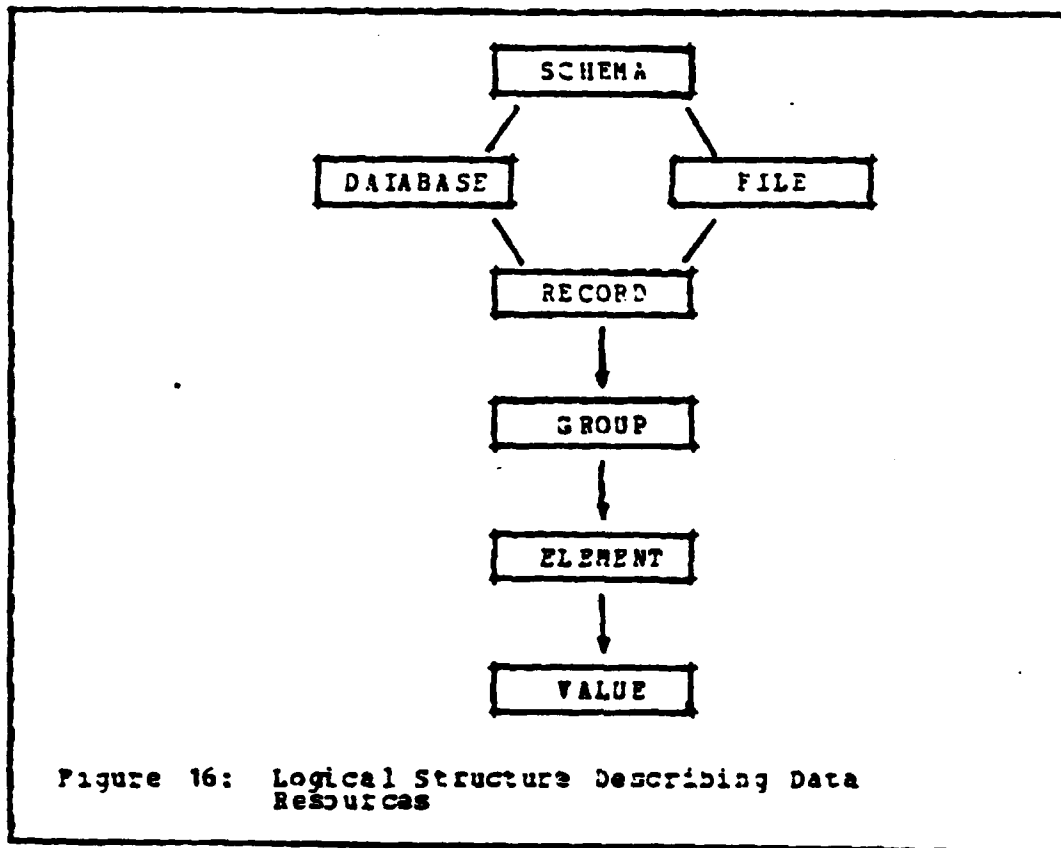
TABLE 2

File Entity Attributes

File name	Format (seq, random, bin)
Locations	Access control
Size (in bytes)	Access security protection

Relationships between data entities are also a function of anticipated usage. One possible network structure relating data entities is shown in Figure 16. This structure implemented in the appropriate DBMS would allow the following queries to be answered via conventional query languages and processors:

1. "List the record layout (all data elements and keys) for the IDA Job Order Reference File."
2. "What locations in the network contain the Purchase Requisition File?"
3. "What files and databases contain the data element FEDERAL-STOCK-NUMBER?"
4. "Where can I order PART-NUMBER = 'POOO345AKF'? "



Hardware entities and attributes have been specified as part of the Configuration Management System (CMS) [SPLICE 81] and a selective sample is shown in Table 3. One valuable use of the DDS regarding hardware resources is in displaying topological information about all or part of the SPLICE network. One logical structure which might provide this capability within a conventional DBMS environment is given in Figure 17. Possible queries might include:

1. "List the number of terminals within each LAN."
2. "Which LANs have database processors?"

TABLE 3

Selected Hardware Entities and Attributes

Entities

Processing system	Concentrators
Secondary storage	Terminals
Communications system	LAN I/O peripherals

Attributes

Type	Features
Model	Description
Model number	Docu. references
Serial number	Usage by site
Mfger's number	Cost
Source	Maintenance activity

Recommended software entities and attributes [SPLICE 81] are summarized in Table 4. One possible logical structure for software, transaction, and report entities is given in Figure 18. The advantage

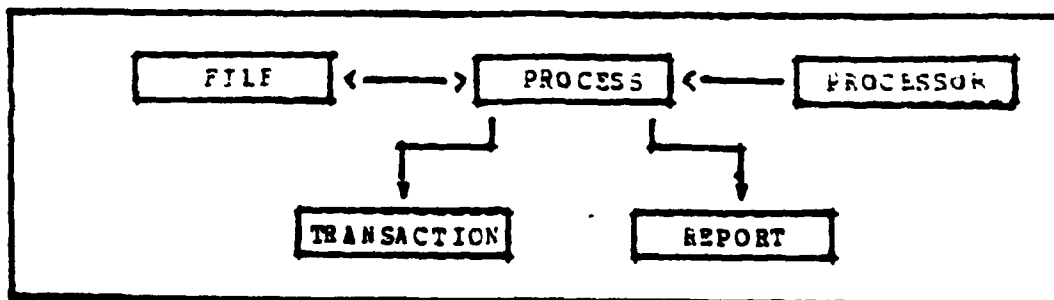
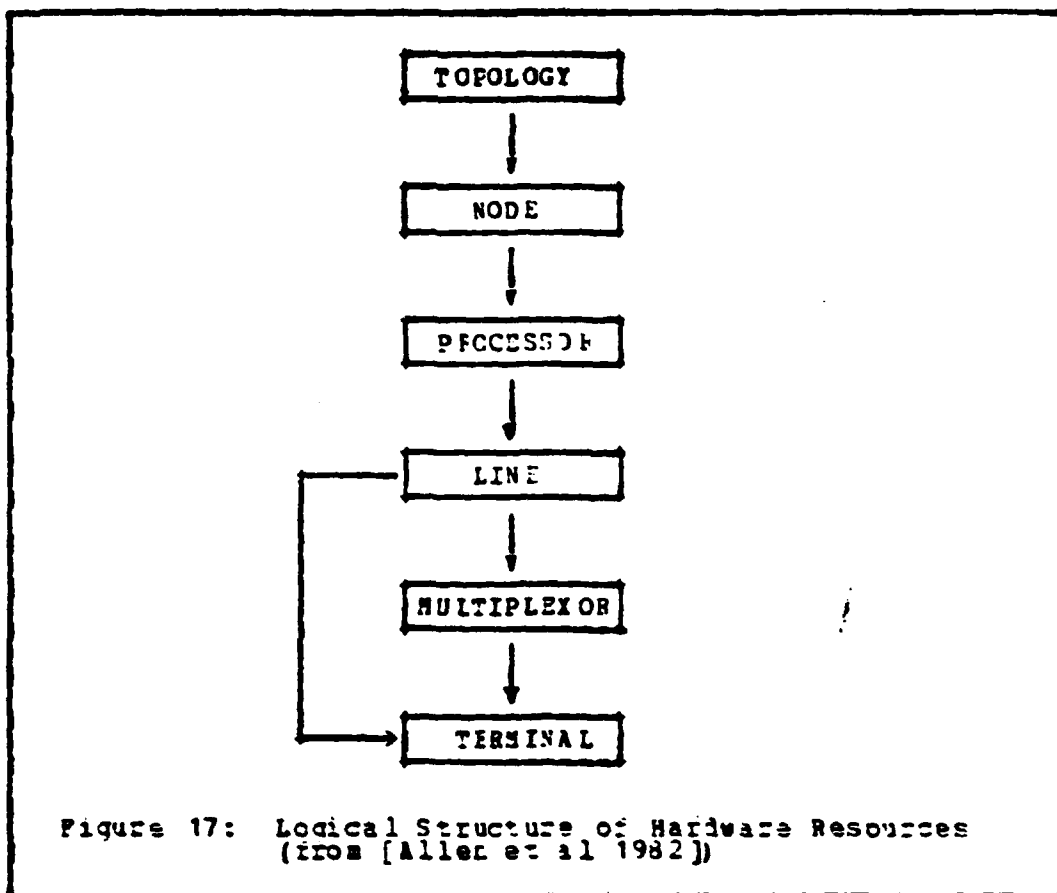


Figure 18: Logical Structure for Software, Transactions, and Report Resources

TABLE 4

Selected Software Entities and Attributes

Entities

Operating system
Operational support system
Environmental system
Application software

Attributes

Program-id
Revision number
Revision date
Date compiled
Type of compiler
Patch level
Change level
License
Date released
Product number
Source
Features
Documentation
Usage
Cost
Maintenance activity

of this kind of structure is that one can extract data flow information from it relatively easily. Thus a query to the effect of "construct a data flow analysis (input files, modules/transactions, output files, reports) for the APADE system" should be reasonable to implement. Inclusion of these resources in the DDS contributes greatly to the viability of the DDS as a systems analysis and design tool.

Other documents besides reports can be catalogued in the DDS as well. Table 5 summarizes entities and attributes for various kinds of documents [SPLICE 81].

Personnel is another resource which can be included in the DDS. Information about users, account numbers and access authority might be germane as well as data about programmers, analysts, DBAs and programs/systems which they are responsible for. Relationships between users and other resources can occur at many levels so there is no attempt to provide any sample structure here. This, again, will be a function of the intended usage of the DDS.

TABLE 5

Document/Report Attributes

Name	Source
Number	Feature
Product number	Description
Release date	Quantity
Revision number	Cost

DISTRIBUTION OF THE DDS

Strongly associated with the issue of logical structure is how to distribute the DDS within the network. The DDS itself consists of one

or more databases and so is subject to distribution just like any other database. It should be noted here that, in general, distribution can occur within, as well as across, local area networks. With SPLICE, intra-LAN distribution will be minimized since database management is to be centralized within each LAN [Schneidewind 82]. Databases will be distributed across LANs, however, and the DDS may be as well.

The major problem in connection with distributing databases and, in particular with distributing the dictionary/directory, is what degree of distribution to implement. Two distinct approaches are generally considered:

1. dividing a database into partitions and locating the partitions at the nodes where they are most likely to be used, or
2. replicating databases and putting copies at the nodes where they are needed.

Partitioning saves secondary storage usage but increases access/communication costs if the partition is suboptimal (i.e., if the demand and supply nodes are frequently different). Replication insures access efficiency but at the cost of increased secondary storage usage. Furthermore, replication introduces the problem of multiple update consistency.

Consider, for example, the hypothetical situation where a supply officer at some NSC has run out of a particular part and wants to order 100 more. He might issue the appropriate query equivalent to "Find all SITES where PART-NO='01037AX' and QTY-ON-HAND 100". The DDS in conjunction with the data management module must respond to this

request. The efficiency of this response depends on how the DDS is distributed.

One possibility might be that each LAN has its own dictionary/directory (D/D) in which case it would be necessary to poll each LAN to see whether it satisfied the query. This is the fully partitioned case and it is clearly costly in terms of communications traffic when the local D/D is unable to satisfy a query.

The fully replicated case would involve a copy of a global D/D being stored within each LAN. In this situation, the query could be satisfied without using the Defense Data Network at all. The problem, of course, is that every change to the D/D must be reflected at each LAN which is again a communication-intensive operation.

A hybrid solution is to store global D/Ds at a very few nodes. Whenever the local D/D cannot provide the information requested, it queries the global D/D at the most proximate site. The question then becomes "where to locate the global D/D copies?" With regard to the kind of query posited in this example, a good choice for locating the global DDS in the SPLICE system might be the Inventory Control Points located at ASO Philadelphia and SPCC Mechanicsburg.

Because of the rather severe tradeoffs involved with a strictly partitioned or strictly replicated approach, some hybrid implementation is most likely. As a result, a distributed DDS will contain both global and local components. This argues for a global DDS logical design which is conducive to partitioning into local components as criteria for this partitioning (perhaps based on functional usage) are determined.

ENVIRONMENTAL DEPENDENCY

Environmental dependency refers to the level of integration of the DDS with other functional modules of the SPLICE system. Earlier, we mentioned that even though the scope of the DDS transcends data management, it is still appropriate to position the DDS within the data management functional module. The next problem to address is the degree to which the DDS should be coupled to a DBMS.

There are three relationships which can exist between a DDS and DBMS [Allen et al. 82]: independent, DBMS-application, and embedded. The independent approach refers to a DDS which does not rely at all upon a particular DBMS. The databases which comprise the DDS are not created or manipulated by a DBMS and interfaces with existing DBM's are generally minimal. The advantage is that the DDS can be used in environments where many different DBMS's operate. The disadvantage is that software capabilities must be provided to perform the data manipulation operations which would otherwise be subsumed by the DBMS.

The DBMS-application approach implies that the DDS is just another database created and manipulated by a particular DBMS. Data manipulation operations are performed by the DBMS but the DBMS D/D and the DDS are separate entities. The advantage is that the full range of DBMS data manipulation features (query processing, report generation, etc.) are available for DDS purposes. The disadvantage is that the DDS is dependent upon a particular DBMS environment.

The embedded approach implies that the DDS is an integral component of the DBMS. All data-directed operations are channeled through the D/D and no other D/Ds exist. The trend in recent years has been towards this fully integrated approach in centralized environments. The

disadvantage of integrated DDSs is that they are limited by DBMS vendor capabilities which may not be sufficient for user needs.

The approach likely to be most suitable for the SPLICE DDS is the DBMS-application approach. A totally independent DDS has certain desirable properties but requires data manipulation capabilities equivalent to those of a DBMS. This would entail a large amount of software development to provide facilities already existing in available DBMS packages.

The fully integrated DDS is also appealing but current integrated systems are designed for centralized environments and are strongly data-oriented rather than resource-oriented. It is not clear whether they can be adapted to accommodate the expanded scope of the SPLICE DDS.

The advantage of the DBMS-application approach is that a customized D/D can be designed and implemented (perhaps integrating logical structures similar to those shown in Figures 16, 17, and 18) which meets the special needs of the SPLICE information resource environment. Furthermore, conventional DBMS features like data description languages, query processors, report generators, and security mechanisms provide ready-made data description, manipulation, and control capabilities for managing the DDS. The primary development effort in this scenario then is one of database design rather than software development.

The major drawback of a DBMS-application DDS is that it might adversely affect the flexibility of its distribution. If the DDS is tied to a particular DBMS, it will not be possible to put any segment or copy of the DDS at a node which does not have the DBMS as well. This can be overcome either by requiring nodes to have uniform DBMS

capabilities or by centralizing a global D/D with few local components. If neither of these alternatives is feasible within SPLICE, then it may be necessary to reconsider the DDS as DBMS-independent instead.

COMPUTER ARCHITECTURE

Consistent with the approach described earlier of starting with the logical design of the system and using it to determine the physical implementation, the design characteristics of LANOS determine the computer architecture of the system and not vice versa. Therefore, in order to support a bus-oriented LAN, the following computer architecture is needed for each installation of the LAN:

A. Local Area Networks

- Topology: Bus
- Access Method: Token bus to allow predictable response times
- Speed: 50M bits/sec. (Hyperchannel, (NSC82))
- Message transmission and acknowledgement
- CRC error checking

B. Minicomputers

- Number: 3 per LAN installation
- Word length: 32 bits
- Logical address space: 24 bits = 16M bytes
- Physical address space: 512K bytes/memory bank
- RAM: Minimum of one 512K byte bank to maximum of 32 per processor
- Cycle time: 200 ns.
- Addressing modes: Direct

Base register: For program relocation

Indirect: For subroutine call return

Indexed: For table manipulation

- Stack for preservation of context switch data
- 16 general purpose registers for PCR, IR, indexing, base registers, indirect addressing and arithmetic
- DMA channels associated with LAN adaptor/buffers operated at 5M bytes/sec
- Disk controllers and disk unites operating at 5M bytes/sec
- Instruction format:

Operation Code	Address Mode	Memory	
		Bank Address	Byte Address
6 bits	2 bits	5 bits	19 bits

- Vector interrupt: To minimize I/O overhead and provide for fast context switch and for servicing interrupts generated from interprocessor communication (start of message input) and completion of message output and for responding to disk completion events. (See Figure 19 for vector interrupt table).

C. Microcomputers

- Several MC68000 class microcomputers for each LAN to support resource allocation and communication tasks
- With a bus operating at a maximum speed of 50M bits/sec (5M bytes/sec), a memory cycle time of 200 ns., a disk channel

speed of 5M bytes/sec, and a DMA speed of 5M bytes/sec, were chosen to keep pace with the bus speed.

CONCLUSIONS AND RECOMMENDATIONS

Operating System

As stated in the Operating System Design objectives section, the objectives of the LANOS are two fold: (1) fast processing and (2) reduction of operating system overhead. It is believed that the design accomplishes these objectives for the following reasons:

- *There is minimum use of the Executive in each processor. That is, the functional modules accomplish their assigned tasks autonomously, without constant need to coordinate with the Executive. In addition, each functional module does as much of the transaction processing as possible when it has possession of a transaction message, thus reducing the amount of functional module interaction and consequent message exchange.
- *The message-based interprocess and interprocessor communication system, working on a software and hardware interrupt basis, provides an asynchronous message communication system which eliminates the message setup time required in many operating systems which employ links, such as in DEMOS (Baskett 77).
- *Many critical resources are dedicated, thus eliminating resource allocation and deallocation time.
- *Multiple computers are used for the major functional modules, thus allowing for a high degree of parallelism.
- *A high speed bus, with matching memory, DMA and disk channel speeds, provide the hardware speed necessary to accomplish and desired objectives.

The disadvantages of the design are possible underutilization of resources as a result of the dedication of resources policy.

Also, there are few examples of successful distributed network operating systems.

The proposed design is, to a large extent, untested. A good deal of simulation will have to be performed in order to test various design assumptions and, in particular, to check the performance which can be achieved with regard to effective bus transfer rate, message transfer time and user response time.

As far as future expansion and enhanced use of the system is concerned, each processor can be expanded with respect to RAM and disk capacity, thus allowing faster functional module execution by providing more buffer and work space per module and increased throughput by providing more storage capacity to support additional communicating functional module pairs. Additional processors could be added to the bus system for performing multiprocessing of the sub-modules of a given functional module or to provide additional functional modules for supporting new applications. The primary limitation on future expansion capabilities will be the effective bus speed which can be sustained as processors are expanded and added to the system.

Dictionary/Directory System

Our objective has been to motivate and substantiate the need for information resource management within the SPLICE system and suggest a means for achieving it. This need is seen as existing on at least two levels:

1. the operating system, the essential function of which is to manage and allocate system resources, and

2. the end-user, who needs to know what information resources exist, where they are located, and how to access them.

A dictionary/directory system is suggested as a vehicle for satisfying both levels although most of the previous discussion was aimed at end-user considerations. The DDS should be resource-oriented rather than just data-oriented if it is to satisfy the above requirements. Since the DDS consists of databases, it is nevertheless appropriate to place it under the aegis of the data management functional module. Several logical structures were discussed as being relevant to the SPLICE DDS design and it was suggested that the DDS be implemented as a DBMS-application database in order to reduce redundant software development.

This study leaves several questions and issues to be explored further:

1. Requirements analysis: this involves a more detailed specification of the kinds of information that should be in the DDS (for both the OS and end-users) and the usage which the DDS will receive; at the end-user level, this may require the determination of resource description standards;
2. DDS design: From the requirements analysis, it will be necessary to devise DDS schemas and determine the local and global components of the database(s);
3. DDS interface: given a DDS design, the interfaces to other SPLICE functional modules must be determined; this is particularly critical at the OS level;

4. Specification of the database/file server: given the role of a DDS, it will be necessary to continue developing specifications for the data management module; in particular, this must be done for the database/file server.

REFERENCES

- [Akkoyunlu 74] Akkoyunlu, Eralp, "Interprocess Communication Facilities for Network Operating Systems", *Computer*, Vol. 7, No. 6, June 1974, pp. 46-55.
- [Allen 82] Allen, F. W., Loomis, M. E. S. and Mannino, M. W., "The Integrated Dictionary/Directory System", *Computing Surveys*, June 1982.
- [Bachman 78] Bachman, Charles and Mike Canepa, "The Session Control Layer of an Open System Interconnection", *Proceedings of Fall COMPCON*, September 1978, pp. 150-156.
- [Bartlett 78] Bartlett, Joel F., "A Non-Stop* Operating System", *Proceedings of the Eleventh International Conference on System Sciences* Vol. III, 1978, pp. 103-117.
- [Baskett 77] Baskett, Forest, et al., "Task Communication in DEMOS", *Proceedings of the Sixth Symposium on Operating Systems Principles*, *ACM Operating Systems Review*, Vol. 11, No. 5, November 1977, pp. 23-31.
- [Boebert 78] Boebert, W.E., et al., "Decentralized Executive Control in Distributed Computer Systems", *Proceedings of COMPSAC 78*, IEEE Computer Society, November 1978, pp. 254-258.
- [Calingaert 82] Calingaert, Peter, Operating System Elements, Prentice-Hall, 1982.
- [Cheriton 79] Cheriton, David R., "Thoth, a Portable Real-Time Operating System", *Communications of the ACM*, Vol. 22, No. 2, February 1979, pp. 105-115.
- [Coffman 71] Coffman, E. G., Jr., M. Elphick and A. Shoshani, "System Deadlocks", *Computing Surveys*, Vol. 13, No. 2, June 1971, pp. 67-78.
- [Curtice 81] Curtice, R. M. "Data Dictionaries: An Assessment of Current Practice and Problems", *VLDB #7 Proceedings*, 1981, pp. 564-570.
- [Deitel 82] Deitel, H. M., An Introduction to Operating Systems, Addison-Wesley, 1982.
- [Donnelley 79] Donnelley, Jed, "Components of a Network Operating System", 4th Conference on Local Computer Networks, IEEE Computer Society, October 1979, pp. 1-12.
- [Guillemont 82] Guillemont, Mare, "The Chorus Distributed Operating System: Design and Implementation", *Local Computer Networks*, Ravasio, et al., (eds.), North-Holland Publishing Co., pp. 207-223.

*"Non-Stop" is a trademark of Tandem Computers, Inc.

- [Jensen 81] Jensen, Douglas E. "Distributed Control, Distributed Systems - Architecture and Implementation", B.W. Tanspion ed.) Springer - Verlag, 1981, pp. 175-190.
- [Jones 79] Jones, Anita K., et al., "StarOS a Multiprocessor Operating System for the Support of Task Forces", Proceedings of the Seventh Symposium on Operating Systems Principles, ACM, December 1979, pp. 117-127.
- [Kuhns 79] Kuhns, Richard C. and Marc C. Shoquist, "A Serial Data Bus System for Local Processing Networks", Digest of Papers, Spring 79 COMPCON, February 1979, pp. 266-271.
- [Metcalf 76] Metcalfe, Robert M. and David R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, Vol. 19, No. 7., July 1976, pp. 395-404.
- [Moore 82] Moore, Lee C., et al., "Design and Implementation of a Local Network Message Passing Protocol", 7th Conference on Local Computer Networks, Computer Society Press, October 1982, pp. 70-74.
- [Nadkarni 83] Nadkarni, Ashok V., et al., "Performance of Some Local Area Network Technologies", Digest of Papers, Spring COMPCON, IEEE Computer Society, March 1983, pp. 137-141.
- [NSC82] Network Systems Corporation, HYPERchannel/TM, Systems Description Manual, January 1982.
- [Ousterhout 80] Ousterhout, John K. et al., "Medusa - An Experiment in Distributed Operating System Structure", Communications of the ACM, Vol. 23, No. 2, February 1980, pp. 92-104.
- [Parker 83] Parker, Richard and Sydney F. Shapiro, "Untangling Local Area Networks", Computer Design, March 1983, pp. 159-172.
- [Pevovar 82] Pevovar, Ed and Brian McGann, "Sorting through the LAN Morass", Digital Design, Vol. 12, No. 11, November 1982, pp. 54-62.
- [Saltzer 81] Saltzer J.H., et al., "Why a Ring?", Seventh Data Communications Symposium, ACM SIGCOMM Computer Communications Review, Vol. 11, No. 14, October 1981, pp. 211-217.
- [Saltzer 82] Saltzer, Jerome H., "On the Naming and Birding of Network Destinations", Local Computer Networks, Ravasio, Piercarlo, et al. (eds.), North-Holland Publishing Co., 1982, pp. 311-317.
- [Salwen 83] Salwen, Howard C. "In Praise of Ring Architecture for Local Area Networks", Computer Design, March 1983, pp. 183-192.
- [Schneidewind 82] Schneidewind, N., "Functional Design of a Local Area Network for the SPLICE Environment", NPS-54-82-003, Naval Postgraduate School, Monterey, CA, December 1982.

- [Schneidewind 83] Schneidewind, Norman F., "Functional Approach to the Design of a Local Network: A Naval Logistics System Example", Digest of Papers, Spring Compcon 83, IEEE Computer Society, March 1983, pp. 197-202.
- [Shoch 78] Shock, J.F., "Internetwork Naming, Addressing and Routing", Proceedings of COMPCON Fall 78, IEEE Computer Society, pp. 72-79.
- [Solomon 79] Solomon, Marvin H. and Raphael A. Finkel, "The Roscoe Distributed Operating System", Proceedings of the Seventh Symposium on Operating Systems Principles, ACM, December 1979, pp. 108-114.
- [SPLICE 81] Fleet Material Office, Department of the Navy, Doc. No. F94LO-9260-001-SS-SU01, SPLICE SYSTEM SPECIFICATIONS, 2 February 1981.
- [Stuck 83] Stuck, Bart W., "Calculating the Maximum Mean Data Rate in Local Area Networks", COMPUTER, Vol. 16, No. 5, May 1983, pp. 72-76.
- [Teichroew 77] Teichroew, D. and Hershey, E., "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, Vol. SE3-No. 1, January 1977, pp. 41-48.
- [Troppe 81] Troppe, Carl, Local Computer Network Technologies, Academic Press, pp. 7-18.
- [Walden 72] Walden, David C., "A System for Interprocess Communication in a Resource Sharing Computer Network", Communication of the ACM, Vol. 15, No. 4, April 1972, pp. 221-230.
- [Watson 81a] Watson, Richard W. "Identifiers (Naming) in Distributed Systems", Distributed System - Architecture and Implementation B.W. Lampson (ed.), Springer - Verlag, 1981, pp. 191-210.
- [Watson 81b] Watson, Richard W., "Distributed System Architecture Model", Ibid, pp. 10-43.
- [Wood 82] Wood, B.J. et al., "A Local-Area Network Architecture Based on Message-Passing Operating System Concepts", 7th Conference on Local Computer Networks, Computer Society Press, October 1982, pp. 59-69.
- [Wulf 74] Wulf, W., et al., "HYDRA: The Kernel of a Multiprocessor Operating System", Communications of the ACM, Vol. 17, No. 6, June 1974, pp. 337-344.
- [Yuen 72] Yuen, M.L.T., et al., "Traffic Flow in a Distributed Loop Switching System", Jerome Fox (ed.), Proceedings of the Symposium on Computer Communications Networks and Teletraffic, Polytechnic Press of the Polytechnic Institute of Brooklyn, 1972, pp. 29-58.

DISTRIBUTION LIST

Number of Copies

Lieutenant Commander Steve Bristow
Navy Management System Support Office
NAS
Norfolk, VA 23464

LCDR Ted Case
Fleet Material Support Office
Code 94L
Mechanicsburg, PA 17055

Professor Dan Dolk
Code 54Dk
Administrative Sciences Department
Naval Postgraduate School
Monterey, CA 93943

Commander Dana Fuller
Commander, Naval Supply Systems Command
Code 0415A
Washington, D.C. 20376

Dr. Harvey A. Freeman
Architecture Technology
P.O. Box 24344
Minneapolis, MN 55424

Captain Chuck Giffried
COMNAVAIRPAC
Code 40
NAS
San Diego, CA 92135

Colonel Heidi B. Heiden
DDN Program Manager
Defense Communications Agency
Attn: DDN PMO (Code B615)
8th & S. Courthouse Roads
Washington, D.C. 20305

Professor Carl Jones
Code 54Js
Administrative Sciences Department
Naval Postgraduate School
Monterey, CA 93943

Ms. Tan Tahn Joo
Dy Head, Software Engineering Department
Information Engineering Centre
System Computer Organisation
Ministry of Defense
Minden Road, Singapore 1024

1

1

1

1

1

1

1

1

1

	<u>Number of Copies</u>
Professor Jack LaPatra Code 54Lp Administrative Sciences Department Naval Postgraduate School Monterey, CA 93943	1
Professor Norm Lyons Code 54Lb Administrative Sciences Department Naval Postgraduate School Monterey, CA 93943	1
Mr. Steve Oxman c/o SHAPE Technical Center United States Research and Development Coordinating Officer APO NY 09159	1
Professor Norman F. Schneidewind Code 54Ss Administrative Sciences Department Naval Postgraduate School Monterey, CA 93943	30
Ms. Mary Willoughby P.O. Box 94 Mendocino, CA 95460	1
Administrative Sciences Department Code 54 Naval Postgraduate School Monterey, CA 93943	1
Computer Center Library Code 0141 Naval Postgraduate School Monterey, CA 93943	1
Computer Science Department Code 52 Naval Postgraduate School Monterey, CA 93943	1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	4

Number of Copies

Office of Research Administration
Code 012A
Naval Postgraduate School
Monterey, CA 93943

1

